FUNDAMENTOS DE BASES DE DATOS

Cuarta edición

FUNDAMENTOS DE BASES DE DATOS

Cuarta edición

Abraham Silberschatz
Bell Laboratories

Henry F. Korth
Bell Laboratories

S. Sudarshan
Instituto Indio de Tecnología, Bombay

Traducción
FERNANDO SÁENZ PÉREZ
ANTONIO GARCÍA CORDERO
CAROLINA LÓPEZ MARTÍNEZ
LUIS MIGUEL SÁNCHEZ BREA
OLGA MATA GÓMEZ
M.ª VICTORIA GONZÁLEZ DEL CAMPO RODRÍGUEZ BARBERO
Universidad Complutense de Madrid

Revisión técnica
LUIS GRAU FERNÁNDEZ
Universidad Nacional de Educación a Distancia



MADRID • BUENOS AIRES • CARACAS • GUATEMALA • LISBOA • MÉXICO
NUEVA YORK • PANAMÁ • SAN JUAN • SANTAFÉ DE BOGOTÁ • SANTIAGO • SÃO PAULO
AUCKLAND • HAMBURGO • LONDRES • MILÁN • MONTREAL • NUEVA DELHI • PARÍS
SAN FRANCISCO • SIDNEY • SINGAPUR • ST. LOUIS • TOKIO • TORONTO

FUNDAMENTOS DE BASES DE DATOS. Cuarta edición

Traducido de la cuarta edición en inglés de
Database System Concepts

ISBN: 0-07-228363-7

ISBN: 84-481-3654-3
Depósito legal: M.

Editora: Concepción Fernández Madrid
Editora de mesa: Susana Santos Prieto
Cubierta: DIMA
Compuesto en FER
Impreso en:

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

En memoria de mi padre, Joseph Silberschatz
y de mis abuelos Stepha y Aaron Resenblum.

Avi Silberschatz

A mi esposa, Joan,
mis hijos, Abigail y Joseph,
y mis padres, Henry y Frances

Hank Korth

A mi esposa, Sita,
mi hijo, Madhur,
y mi madre, Indira.

S. Sudarshan

CONTENIDO BREVE

CONTENIDO

PARTE OCTAVA: ESTUDIO DE CASOS

PREFACIO

LA gestión de bases de datos ha evolucionado desde una aplicación informática especializada hasta una parte esencial de un entorno informático moderno y, como resultado, el conocimiento acerca de los sistemas de bases de datos se ha convertido en una parte esencial en la enseñanza de la informática. En este libro se presentan los conceptos fundamentales de la administración de bases de datos. Estos conceptos incluyen aspectos de diseño de bases de datos, lenguajes de bases de datos e implementación de sistemas de bases de datos.

Este libro está orientado a un primer curso de bases de datos para niveles técnicos y superiores. Además del material básico para un primer curso, el texto también contiene temas que pueden usarse como complemento del curso o como material introductorio de un curso avanzado.

En este libro se asume que se dispone de los conocimientos elementales sobre estructuras de datos básicas, organización de computadoras y un lenguaje de programación de alto nivel (tipo Pascal). Los conceptos se presentan usando descripciones intuitivas, muchas de las cuales están basadas en el ejemplo propuesto de una empresa bancaria. Se tratan los resultados teóricos importantes, pero se omiten las demostraciones formales. Las notas bibliográficas contienen referencias a artículos de investigación en los que los resultados se presentaron y probaron, y también referencias a material para otras lecturas. En lugar de demostraciones, se usan figuras y ejemplos para sugerir por qué se espera que los resultados en cuestión sean ciertos.

Los conceptos fundamentales y algoritmos tratados en este libro se basan habitualmente en los que se usan en la actualidad en sistemas de bases de datos existentes, comerciales o experimentales. Nuestro deseo es presentar estos conceptos y algoritmos como un conjunto general que no esté ligado a un sistema de bases de datos particular. En la Parte 8 se discuten detalles de sistemas de bases de datos comerciales.

En esta cuarta edición de *Fundamentos de bases de datos* se ha mantenido el estilo global de las primeras tres ediciones, a la vez que se ha tenido en cuenta la evolución de la gestión de bases de datos. Se han añadido varios capítulos nuevos para tratar nuevas tecnologías. Cada capítulo se ha corregido y la mayoría se ha modificado ampliamente. Se describirán los cambios con detalle en breve.

ORGANIZACIÓN

El texto está organizado en ocho partes principales más dos apéndices:

- **Visión general** (Capítulo 1). En el Capítulo 1 se proporciona una visión general de la naturaleza y propósito de los sistemas de bases de datos. Se explica cómo se ha desarrollado el concepto de sistema de bases de datos, cuáles son las características usuales de los sistemas de bases de datos, lo que proporciona al usuario un sistema de bases de datos y cómo un sistema de bases de datos se comunica con los sistemas operativos. También se introduce una aplicación de bases de datos de ejemplo: una empresa bancaria que consta de muchas sucursales. Este ejemplo se usa a lo largo de todo el libro. Este capítulo es histórico, explicativo y motivador por naturaleza.
- **Modelos de datos** (Capítulos 2 y 3). En el Capítulo 2 se presenta el modelo entidad-relación. Este modelo proporciona una visión de alto nivel de los resultados de un diseño de base de datos y de los problemas que se encuentran en la captura de la semántica de las aplicaciones realistas que contienen las restricciones de un modelo de datos. El Capítulo 3 se centra en el modelo de datos relacional, tratando la relevancia del álgebra relacional y el cálculo relacional.
- **Bases de datos relacionales** (Capítulos 4 al 7). El Capítulo 4 se centra en el lenguaje relacional orientado al usuario de mayor influencia: SQL. El Capítulo 5 cubre otros dos lenguajes relacionales, QBE y Datalog. En estos dos capítulos se describe la manipulación de datos: consultas, actualizaciones, inserciones y borrados. Los algoritmos y las cuestiones de diseño

se relegan a capítulos posteriores. Así, estos capítulos son adecuados para aquellas personas o para las clases de nivel más bajo en donde se desee aprender qué es un sistema de bases de datos, sin entrar en detalles sobre los algoritmos internos y estructuras que contienen.

En el Capítulo 6 se presentan las restricciones desde el punto de vista de la integridad de las bases de datos. En el Capítulo 7 se muestra cómo se pueden usar las restricciones en el diseño de una base de datos relacional. En el Capítulo 6 se presentan la integridad referencial; mecanismos para el mantenimiento de la integridad, tales como disparadores y asertos, y mecanismos de autorización. El tema de este capítulo es la protección de las bases de datos contra daños accidentales y daños intencionados.

En el Capítulo 7 se introduce la teoría del diseño de bases de datos relacionales. Se trata la teoría de las dependencias funcionales y la normalización, con énfasis en la motivación y el significado intuitivo de cada forma normal. También se describe en detalle el proceso de diseño de bases de datos.

- **Bases de datos basadas en objetos y XML** (Capítulos 8 al 10). El Capítulo 8 trata las bases de datos orientadas a objetos. En él se introducen los conceptos de la programación orientada a objetos y se muestra cómo estos conceptos constituyen la base para un modelo de datos. No se asume un conocimiento previo de lenguajes orientados a objetos. El Capítulo 9 trata las bases de datos relacionales de objetos, y muestra cómo la norma SQL:1999 extiende el modelo de datos relacional para incluir características de la programación orientada a objetos, tales como la herencia, los tipos complejos y la identidad de objeto.

  En el Capítulo 10 se trata la norma XML para representación de datos, el cual está experimentando un uso cada vez mayor en la comunicación de datos y en el almacenamiento de tipos de datos complejos. El capítulo también describe lenguajes de consulta para XML.

- **Almacenamiento de datos y consultas** (Capítulos 11 al 14). En el Capítulo 11 se estudian los discos, archivos y estructuras de un sistema de archivos y la correspondencia de datos relacionales y de objetos con un sistema de archivos. En el Capítulo 12 se presentan varias técnicas de acceso a los datos, incluyendo la asociación, los índices de árboles B+ y los índices de archivos en retícula. Los capítulos 13 y 14 tratan los algoritmos de evaluación de consultas y optimización de consultas basados en transformación de consultas preservando la equivalencia.

  Estos capítulos están orientados a personas que desean conocer los componentes de almacenamiento y consulta internos de una base de datos.

- **Gestión de transacciones** (Capítulos 15 al 17). El Capítulo 15 se centra en los fundamentos de un sistema de procesamiento de transacciones, incluyendo la atomicidad de las transacciones, la consistencia, el aislamiento y la durabilidad, y también la noción de secuencialidad.

  El Capítulo 16 se centra en el control de concurrencia y se presentan varias técnicas que aseguran la secuencialidad, incluyendo los bloqueos, las marcas temporales y técnicas optimistas (de validación). Los temas de interbloqueo se tratan también en este capítulo. El Capítulo 17 aborda las técnicas principales para asegurar la ejecución correcta de transacciones a pesar de las caídas del sistema y los fallos de disco. Estas técnicas incluyen el registro histórico, paginación en la sombra, puntos de revisión y volcados de la base de datos.

- **Arquitectura de un sistema de bases de datos** (Capítulos 18 al 20). El Capítulo 18 trata la arquitectura de un sistema informático y en él se describe la influencia de los sistemas informáticos subyacentes en los sistemas de bases de datos. Se discuten los sistemas centralizados, los sistemas cliente-servidor, las arquitecturas paralelas y distribuidas, y los tipos de redes. En el Capítulo 19 se estudian los sistemas de bases de datos distribuidas, revisando los aspectos de diseño de bases de datos, gestión de las transacciones y evaluación y optimización de consultas en el contexto de los sistemas de bases de datos distribuidas. El capítulo también trata aspectos de la disponibilidad del sistema durante fallos y describe el sistema de directorios LDAP.

  En el capítulo 20, acerca de las bases de datos paralelas, se exploran varias técnicas de paralelización, incluyendo paralelismo de E/S, paralelismo entre consultas y en consultas, y paralelismo entre operaciones y en operaciones. También se describe el diseño de sistemas paralelos.

- **Otros temas** (Capítulos 21 al 24). El Capítulo 21 trata el desarrollo y administración de aplicaciones de bases de datos. Los temas incluyen las interfaces de las bases de datos, en particular las interfaces Web, el ajuste de rendimiento, los programas de prueba, la estandarización y los aspectos de las bases de datos en el comercio electrónico. El Capítulo 22 presenta

técnicas de consulta, incluyendo sistemas de ayuda a la toma de decisiones y recuperación de la información. Los temas tratados en el área de la ayuda a la toma de decisiones incluyen las técnicas de procesamiento analítico interactivo (OLAP, Online Analytical Processing), el soporte de SQL:1999 para OLAP, recopilación de datos y almacenes de datos. El capítulo también describe técnicas de recuperación de información para la consulta de datos textuales, incluyendo técnicas basadas en hipervínculos usadas en los motores de búsqueda Web.

El Capítulo 23 trata tipos de datos avanzados y nuevas aplicaciones, incluyendo datos temporales, datos espaciales y geográficos, datos multimedia, y aspectos de la gestión de las bases de datos móviles y personales. Finalmente, el Capítulo 24 trata el procesamiento avanzado de transacciones. Se estudian los monitores de procesamiento de transacciones, los sistemas de transacciones de alto rendimiento, los sistemas de transacciones de tiempo real, y los flujos de datos transaccionales.

- **Estudios de casos** (Capítulos 25 al 27). En esta parte presentamos estudios de casos de tres sistemas de bases de datos comerciales: Oracle, IBM DB2 y Microsoft SQL Server. Estos capítulos esbozan características únicas de cada uno de los productos y describen su estructura interna. Proporcionan una gran cantidad de información interesante sobre los productos respectivos, y ayudan a ver cómo las diferentes técnicas de implementación descritas en las partes anteriores se usan en sistemas reales. También se tratan aspectos prácticos en el diseño de sistemas reales.
- **Apéndices en línea.** Aunque la mayoría de las aplicaciones de bases de datos modernas usen, bien el modelo relacional o bien el modelo orientado a objetos, los modelos de datos de redes y jerárquico están en uso todavía. En beneficio de los lectores que deseen aprender estos modelos de datos se proporcionan apéndices que describen los modelos de redes y jerárquico, en los Apéndices A y B, respectivamente. Los apéndices sólo están disponibles en Internet (http://www.bell-labs.com/topic/books/db-book).

  El Apéndice C describe el diseño avanzado de bases de datos relacionales, incluyendo la teoría de dependencias multivaloradas ¿Multivaluadas?, las dependencias de reunión y las formas normales de proyección-reunión y dominio-clave. Este apéndice es útil para quienes deseen el tratamiento del diseño de bases de datos relacionales en más detalle, y para profesores que deseen explicarlo en sus asignaturas. Este apéndice está también sólo disponible en Internet, en la página Web del libro.

LA CUARTA EDICIÓN

La producción de esta cuarta edición se ha guiado por muchos comentarios y sugerencias referidos a las ediciones anteriores, junto con las propias observaciones en la enseñanza en el IIT de Bombay, y por el análisis de las direcciones que la tecnología de bases de datos está tomando.

El procedimiento básico fue reescribir el material en cada capítulo, actualizando el material más antiguo, añadiendo discusiones en desarrollos recientes en la tecnología de bases de datos, y mejorando las descripciones de los temas que los estudiantes encontraron difíciles de comprender. Cada capítulo tiene ahora una lista de términos de repaso, que pueden ayudar a asimilar los temas clave tratados en cada capítulo. Se ha añadido también una nueva sección al final de la mayoría de los capítulos que proporciona información sobre herramientas software referidas al tema del capítulo. También se han añadido nuevos ejercicios y se han actualizado las referencias.

Se ha incluido un nuevo capítulo que trata XML, y tres capítulos de estudio de los sistemas de bases de datos comerciales líderes: Oracle, IBM DB2 y Microsoft SQL Server.

Los capítulos se han organizado en varias partes y se han reorganizado los contenidos de varios de ellos. En beneficio de aquellos lectores familiarizados con la tercera edición se explican a continuación los principales cambios.

- **Modelo entidad-relación.** Se ha mejorado el tratamiento del modelo entidad-relación (E-R). Se han añadido nuevos ejemplos y algunos se han cambiado para dar una mejor intuición al lector. Se ha incluido un resumen de notaciones E-R alternativas, junto con un nuevo apartado sobre UML.
- **Bases de datos relacionales.** El tratamiento de SQL en el Capítulo 4 ahora se refiere al estándar SQL:1999, que se aprobó después de la publicación de la tercera edición de este libro. El

tratamiento de SQL se ha ampliado significativamente para incluir la cláusula **with**, para un tratamiento ampliado de SQL incorporado y el tratamiento de ODBC y JDBC, cuyo uso ha aumentado notablemente en los últimos años. La parte del capítulo 5 dedicada a Quel se ha eliminado, ya que no se usa ampliamente debido al poco uso que actualmente se hace de este lenguaje. El tratamiento de QBE se ha revisado para eliminar algunas ambigüedades y para añadir el tratamiento de la versión de QBE usada en la base de datos Microsoft Access.

El Capítulo 6 trata ahora de las restricciones de integridad y de la seguridad. El tratamiento de la seguridad, ubicado en la edición anterior en el Capítulo 19, se ha trasladado al Capítulo 6. El Capítulo 6 también trata los disparadores. El Capítulo 7 aborda el diseño de las bases de datos relacionales y las formas normales. La discusión de las dependencias funcionales, ubicada en la edición anterior en el Capítulo 6, se ha trasladado al Capítulo 7. El Capítulo 7 se ha remodelado significativamente, proporcionando varios algoritmos para las dependencias funcionales y un tratamiento extendido del proceso general del diseño de bases de datos. Los axiomas para la inferencia de las dependencias multivaloradas, las formas normales FNRP y FNCD se han trasladado al apéndice.

- **Bases de datos basadas en objetos.** Se ha mejorado el tratamiento de la orientación a objetos del Capítulo 8, y se ha actualizado la discusión de ODMG. Se ha actualizado el tratamiento de las bases de datos relacionales orientadas a objetos del Capítulo 9 y, en particular, el estándar SQL:1999, reemplaza a SQL extendido usado en la tercera edición.
- **XML.** El Capítulo 10, que trata XML, es un nuevo capítulo de la cuarta edición.
- **Almacenamiento, indexación y procesamiento de consultas.** Se ha actualizado el tratamiento del almacenamiento y de las estructuras de archivos del Capítulo 11; este fue el Capítulo 10 en la tercera edición. Muchas características de las unidades de disco y de otros mecanismos de almacenamiento han cambiado en gran medida con el paso de los años, y su tratamiento se ha actualizado correspondientemente. El tratamiento de RAID se ha actualizado para reflejar las tendencias tecnológicas. El tratamiento de diccionarios de datos (catálogos) se ha extendido.

  El Capítulo 12, sobre indexación, incluye ahora el estudio de los índices de mapa de bits; este capítulo fue el Capítulo 11 en la tercera edición. El algoritmo de inserción en árboles B^+ se ha simplificado y se ha proporcionado un pseudocódigo para su examen. La asociación dividida se ha eliminado, ya que no tiene un uso significativo.

  El tratamiento del procesamiento de consultas se ha reorganizado, con el capítulo anterior (Capítulo 12 en la tercera edición) dividido en dos capítulos, uno sobre procesamiento de consultas (Capítulo 13) y otro sobre optimización de consultas (Capítulo 14). Todos los detalles referidos a la estimación de costes y a la optimización de consultas se han trasladado al Capítulo 14, permitiendo al Capítulo 13 centrarse en los algoritmos de procesamiento de consultas. Se han eliminado varias fórmulas detalladas (y tediosas) para el cálculo del número exacto de operaciones de E/S para diferentes operaciones. El Capítulo 14 se presenta ahora con un pseudocódigo para la optimización de algoritmos, y nuevos apartados sobre la optimización de subconsultas anidadas y sobre vistas materializadas.
- **Procesamiento de transacciones.** El Capítulo 15, que proporciona una introducción a las transacciones, se ha actualizado; este capítulo era el Capítulo 13 en la tercera edición. Se han eliminado los tests de secuenciabilidad.

  El Capítulo 16, sobre el control de concurrencia, incluye un nuevo apartado sobre la implementación de los gestores de bloqueo, y otro sobre los niveles débiles de consistencia, que estaban en el Capítulo 20 de la tercera edición. Se ha ampliado el control de concurrencia de estructuras de índices, proporcionando detalles del protocolo cangrejo, que es una alternativa más simple al protocolo de enlace B, y el bloqueo de siguiente clave para evitar el problema fantasma. El Capítulo 17, que trata sobre recuperación, incluye ahora un estudio del algoritmo de recuperación ARIES. Este capítulo trata ahora los sistemas de copia de seguridad remota para proporcionar una alta disponibilidad a pesar de los fallos, característica cada vez más importante en las aplicaciones «24×7».

  Como en la tercera edición, esta organización permite a los profesores elegir entre conceptos de procesamiento de transacciones introductorios únicamente (cubiertos sólo en el Capítulo 15) u ofrecer un conocimiento detallado (basado en los Capítulos 15 al 17).
- **Arquitecturas de sistemas de bases de datos.** El Capítulo 18, que proporciona una visión general de las arquitecturas de sistemas de bases de datos, se ha actualizado para tratar la tecnología actual; esto se encontraba en el Capítulo 16 de la tercera edición. El orden del capí-

tulo de bases de datos paralelas y de los capítulos de bases de datos distribuidas se ha intercambiado. Mientras que el tratamiento de las técnicas de procesamiento de consultas de bases de datos del Capítulo 20 (que fue el Capítulo 16 en la tercera edición) es de primordial interés para quienes deseen aprender los interiores de las bases de datos, las bases de datos distribuidas, ahora tratadas en el Capítulo 19, son un tema más fundamental con el que debería estar familiarizado cualquiera que trabaje con bases de datos.

El Capítulo 19 sobre bases de datos distribuidas se ha rehecho significativamente para reducir el énfasis en la denominación y la transparencia, y para aumentar el tratamiento de la operación durante fallos, incluyendo las técnicas de control de concurrencia para proporcionar alta disponibilidad. El tratamiento del protocolo de compromiso de tres fases se ha abreviado, al tener detección distribuida de interbloqueos globales, ya que no se usa mucho en la práctica. El estudio de los aspectos de procesamiento de consultas se ha trasladado del Capítulo 20 de la tercera edición. Hay un nuevo apartado sobre los sistemas de directorio, en particular LDAP, ya que se usan ampliamente como un mecanismo para hacer disponible la información en una configuración distribuida.

- **Otros temas.** Aunque se ha modificado y actualizado el texto completo, nuestra presentación del material que tiene relación con el continuo desarrollo de bases de datos y las nuevas aplicaciones de bases de datos se tratan en cuatro nuevos capítulos, del Capítulo 21 al 24.

  El Capítulo 21 es nuevo en la cuarta edición y trata el desarrollo y administración de aplicaciones. La descripción de la construcción de interfaces Web para bases de datos, incluyendo servlets y otros mecanismos para las secuencias de comandos para el lado del servidor, es nueva. La sección sobre ajuste de rendimiento, que estaba anteriormente en el Capítulo 19, tiene nuevo material sobre la famosa regla de 5 minutos y sobre la regla de 1 minuto, así como algunos nuevos ejemplos. El tratamiento de la selección de vistas materializadas también es nuevo. El tratamiento de los programas de prueba y de los estándares se ha actualizado. Hay una nueva sección sobre comercio electrónico, centrándose en los aspectos de las bases de datos en el comercio electrónico y un nuevo apartado que trata los sistemas heredados.

  El Capítulo 22, que trata consultas avanzadas y recuperación de la información, incluye nuevo material sobre OLAP, particularmente sobre las extensiones de SQL:1999 para análisis de datos. El estudio de los almacenes de datos y de la recopilación de datos también se ha ampliado en gran medida. El tratamiento de la recuperación de la información se ha aumentado significativamente, en particular en el área de la búsqueda Web. Las versiones anteriores de este material estaban en el Capítulo 21 de la tercera edición.

  El Capítulo 23, que trata tipos de datos avanzados y nuevas aplicaciones, contiene material sobre datos temporales, datos espaciales, datos multimedia y bases de datos móviles. Este material es una versión actualizada del material que se encontraba en el Capítulo 21 de la tercera edición. El Capítulo 24, que trata el procesamiento de transacciones avanzado, contiene versiones actualizadas de los monitores TP, sistemas de flujo de datos, bases de datos en memoria principal y de tiempo real, transacciones de larga duración y gestión de transacciones en múltiples bases de datos, que aparecieron en el Capítulo 20 de la tercera edición.

NOTA PARA EL PROFESOR

El libro contiene tanto material básico como material avanzado, que podría no ser abordado en un único semestre. Se han marcado varios apartados como avanzados, usando el símbolo «**». Estos apartados se pueden omitir, si se desea, sin pérdida de continuidad.

Es posible diseñar cursos usando varios subconjuntos de los capítulos. A continuación se muestran varias posibilidades:

- El Capítulo 5 se puede omitir si los estudiantes no van a usar QBE o Datalog como parte del curso.
- Si la programación orientada a objetos se va a tratar en un curso avanzado por separado, los Capítulos 8 y 9 y el Apartado 11.9 se pueden omitir. Alternativamente, con ellos se puede constituir la base de un curso avanzado de bases de datos orientadas a objetos.
- El Capítulo 10 (XML) y el Capítulo 14 (optimización de consultas) se pueden omitir para un curso introductorio.

- Tanto el tratamiento del procesamiento de transacciones (Capítulos 15 al 17) como de la arquitectura de sistemas de bases de datos (Capítulos 18 al 20) poseen un capítulo de visión de conjunto (Capítulos 15 y 18 respectivamente), seguidos de capítulos más detallados. Se podría elegir usar los Capítulos 15 y 18, omitiendo los Capítulos 16, 17, 19 y 20, si se relegan estos capítulos para un curso avanzado.
- Los Capítulos 21 al 24 son adecuados para un curso avanzado o para autoaprendizaje de los estudiantes, aunque el apartado 21.1 se puede tratar en un primer curso de bases de datos.

Se puede encontrar un modelo de plan de estudios del curso, a partir del texto, en la página inicial Web del libro (véase el siguiente apartado).

PÁGINA WEB Y SUPLEMENTOS PARA LA ENSEÑANZA

Está disponible una página World Wide Web para este libro en el URL:

http://www.bell-labs.com/topic/books/db-book

La página Web contiene:

- Transparencias de todos los capítulos del libro.
- Respuestas a ejercicios seleccionados.
- Los tres apéndices.
- Una lista de erratas actualizada.
- Material suplementario proporcionado por usuarios del libro.

Se puede proporcionar un manual completo de soluciones sólo a las facultades. Para obtener más información sobre cómo obtener una copia del manual de soluciones envíe por favor un correo electrónico a customer.service@mcgraw-hill.com. En los Estados Unidos se puede llamar al 800-338-3987. La página Web de McGraw-Hill para este libro es:

http://www.mcgraw-hill.es/olc/silberschatz

CÓMO CONTACTAR CON LOS AUTORES Y OTROS USUARIOS

Se ha creado una lista de correo electrónico con la que los usuarios de este libro pueden comunicarse entre sí y con los autores. Si desea incorporarse a la lista, por favor mande un mensaje a db-book@research.bell-labs.com, incluyendo su nombre, afiliación, puesto y dirección de correo electrónico.

Nos hemos esforzado para eliminar erratas y problemas del texto, pero, como en los nuevos desarrollos de software, probablemente queden fallos; hay una lista de erratas actualizada accesible desde la página inicial del libro*. Agradeceríamos que se nos notificara cualquier error u omisión del libro que no se encuentre en la lista actual de erratas.

También desearíamos recibir sugerencias sobre la mejora del libro. Damos la bienvenida a aquellas contribuciones a la página Web del libro que pudiesen ser usadas por otros lectores, como ejercicios de programación, sugerencias sobre proyectos, laboratorios y tutoriales en línea, y consejos de enseñanza.

El correo electrónico se deberá dirigir a db-book@research.bell-labs.com. Cualquier otra correspondencia se debe enviar a Avi Silberschatz, Bell Laboratories, Room 2T-310, 600 Mountain Avenue, Murray Hill, NJ 07974, EE.UU.

---

* N. del T. Todas las erratas de la versión original en inglés incluidas en esta página Web en el momento de finalizar la traducción se han corregido en este libro.

AGRADECIMIENTOS

Esta edición se ha beneficiado de los muchos y útiles comentarios que nos han proporcionado los muchos estudiantes que han usado la tercera edición. Además, gran cantidad de personas nos han escrito o hablado acerca del libro, y nos han ofrecido sugerencias y comentarios. Aunque no podemos mencionar aquí a todas, agradecemos especialmente a las siguientes:

- Phil Bernhard, Instituto de Tecnología de Florida; Eitan M. Gurari, Universidad del estado de Ohio; Irwin Levinstein, Universidad Old Dominion; Ling Liu, Instituto de Tecnología de Georgia; Ami Motro, Universidad George Mason; Bhagirath Narahari, Meral Ozsoyoglu, Universidad Case Western Reserve; y Odinaldo Rodríguez, King's College de Londres; que sirvieron como revisores del libro y cuyos comentarios nos ayudaron en gran medida en la formulación de esta cuarta edición.
- Soumen Chakrabarti, Sharad Mehrotra, Krithi Ramamritham, Mike Reiter, Sunita Sarawagi, N. L. Sarda y Dilys Thomas, por su amplia y valiosa realimentación sobre varios capítulos del libro.
- Phil Bohannon, por escribir el primer borrador del Capítulo 10 describiendo XML.
- Hakan Jakobsson (Oracle), Sriram Padmanabbhan (IBM) y César Galindo-Legareia, Goetz Graefe, José A. Blakeley, Kalen Delaney, Michael Rys, Michael Zwilling, Sameet Agarwal, Thomas Casey (todos de Microsoft), por escribir los apéndices sobre los sistemas de bases de datos Oracle, IBM DB2 y Microsoft SQL Server.
- Yuri Breitbart, por su ayuda con el capítulo sobre bases de datos distribuidas; Mark Reiter, por su ayuda con los apartados de seguridad; y Jim Melton, por las aclaraciones sobre SQL:1999.
- Marilyn Turnamian y Nandprasad Joshi, cuya excelente asistencia secretarial fue esencial para terminar a tiempo esta cuarta edición.

La editora fue Betsy Jones. El editor de desarrollo senior fue Kelly Butcher. El director de proyecto fue Jill Peter. El director de marketing ejecutivo fue John Wannemacher. El ilustrador de la portada fue Paul Tumbaugh mientras que el diseñador de la portada fue JoAnne Schopler. El editor de copia fue George Watson. El corrector de pruebas fue Marie Zartman. El productor del material complementario fue Jodi Banowetz. El diseñador fue Rick Noel. El indexador fue Tobiah Waldron.

Esta edición está basada en las tres ediciones previas, así que los autores dan las gracias una vez más a las muchas personas que ayudaron con las tres primeras ediciones, incluyendo a R.B. Abhyankar, Don Batory, Haran Boral, Paul Bourgeois, Robert Brazile, Michael Carey, J. Edwards, Christos Faloutsos, Homma Farian, Alan Fekete, Shashi Gadia, Jim Gray, Le Gruenwald, Yannis Ioannidis, Hyoung-Joo Kim, Henry Korth (padre de Henry F.), Carol Kroll, Gary Lindstrom, Dave Maier, Keith Marzullo, Fletcher Mattox, Alberto Mendelzon, Héctor García-Molina, Ami Motro, Anil Nigam, Cyril Orji, Bruce Porter, Jim Peterson, K.V. Raghavan, Mark Roth, Marek Rusinkiewicz, S. Seshadri, Shashi Shekhar, Amit Sheth, Nandit Soparkar, Greg Speegle y Marianne Winslett. Lyn Dupré editó y corrigió la tercera edición del libro y Sara Strandtman editó el texto de la tercera edición. Greg Speegle, Dawn Bezviner y K. V. Raghavan nos ayudaron a preparar el manual del profesor para ediciones anteriores. La nueva portada es una evolución de las portadas de las tres primeras ediciones. Marilyn Turnamian creó un primer boceto del diseño de la portada para esta edición. La idea de usar barcos como parte del concepto de la portada fue sugerida originalmente por Bruce Stephan.

Finalmente, Sudarshan desearía agradecer a su esposa, Sita, por su amor y apoyo, a su hijo de dos años Madhur por su amor, y a su madre, Indira, por su apoyo. Hank desearía agradecer a su esposa, Joan, y a sus hijos, Abby y Joe, por su amor y comprensión. Avi desearía agradecer a su esposa Haya y a su hijo Aaron por su paciencia y apoyo durante la revisión de este libro.

A.S.

H.F.K.

S.S.

CAPÍTULO

1 INTRODUCCIÓN

UN **sistema gestor de bases de datos** (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. La colección de datos, normalmente denominada **base de datos**, contiene información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto *práctica* como *eficiente*.

Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información. Además, los sistemas de bases de datos deben proporcionar la fiabilidad de la información almacenada, a pesar de las caídas del sistema o los intentos de acceso sin autorización. Si los datos van a ser compartidos entre diversos usuarios, el sistema debe evitar posibles resultados anómalos.

Dado que la información es tan importante en la mayoría de las organizaciones, los científicos informáticos han desarrollado un amplio conjunto de conceptos y técnicas para la gestión de los datos. En este capítulo se presenta una breve introducción a los principios de los sistemas de bases de datos.

1.1. APLICACIONES DE LOS SISTEMAS DE BASES DE DATOS

Las bases de datos son ampliamente usadas. Las siguientes son algunas de sus aplicaciones más representativas:

- *Banca*. Para información de los clientes, cuentas y préstamos, y transacciones bancarias.
- *Líneas aéreas*. Para reservas e información de planificación. Las líneas aéreas fueron de los primeros en usar las bases de datos de forma distribuida geográficamente (los terminales situados en todo el mundo accedían al sistema de bases de datos centralizado a través de las líneas telefónicas y otras redes de datos).
- *Universidades*. Para información de los estudiantes, matrículas de las asignaturas y cursos.
- *Transacciones de tarjetas de crédito*. Para compras con tarjeta de crédito y generación mensual de extractos.
- *Telecomunicaciones*. Para guardar un registro de las llamadas realizadas, generación mensual de facturas, manteniendo el saldo de las tarjetas telefónicas de prepago y para almacenar información sobre las redes de comunicaciones.
- *Finanzas*. Para almacenar información sobre grandes empresas, ventas y compras de documentos formales financieros, como bolsa y bonos.
- *Ventas*. Para información de clientes, productos y compras.
- *Producción*. Para la gestión de la cadena de producción y para el seguimiento de la producción de elementos en las factorías, inventarios de elementos en almacenes y pedidos de elementos.
- *Recursos humanos*. Para información sobre los empleados, salarios, impuestos y beneficios, y para la generación de las nóminas.

Como esta lista ilustra, las bases de datos forman una parte esencial de casi todas las empresas actuales.

A lo largo de las últimas cuatro décadas del siglo veinte, el uso de las bases de datos creció en todas las empresas. En los primeros días, muy pocas personas interactuaron directamente con los sistemas de bases de datos, aunque sin darse cuenta interactuaron con bases de datos indirectamente (con los informes impresos como extractos de tarjetas de crédito, o mediante agentes como cajeros de bancos y agentes de reserva de líneas aéreas). Después vinieron los cajeros automáticos y permitieron a los usuarios interactuar con las bases de datos. Las interfaces telefónicas con los computadores (sistemas de respuesta vocal interactiva) también permitieron a los usuarios manejar directamente las bases de datos. Un llamador podía marcar un número y pulsar teclas del teléfono para introducir información o para seleccionar opciones alternativas, para determinar las horas de llegada o salida, por ejemplo, o para matricularse de asignaturas en una universidad.

La revolución de Internet a finales de la década de 1990 aumentó significativamente el acceso directo del

usuario a las bases de datos. Las organizaciones convirtieron muchas de sus interfaces telefónicas a las bases de datos en interfaces Web, y pusieron disponibles en línea muchos servicios. Por ejemplo, cuando se accede a una tienda de libros en línea y se busca un libro o una colección de música se está accediendo a datos almacenados en una base de datos. Cuando se solicita un pedido en línea, el pedido se almacena en una base de datos. Cuando se accede a un banco en un sitio Web y se consulta el estado de la cuenta y los movimientos, la información se recupera del sistema de bases de datos del banco. Cuando se accede a un sitio Web, la información personal puede ser recuperada de una base de datos para seleccionar los anuncios que se deberían mostrar. Más aún, los datos sobre los accesos Web pueden ser almacenados en una base de datos.

Así, aunque las interfaces de datos ocultan detalles del acceso a las bases de datos, y la mayoría de la gente ni siquiera es consciente de que están interactuando con una base de datos, el acceso a las bases de datos forma una parte esencial de la vida de casi todas las personas actualmente.

La importancia de los sistemas de bases de datos se puede juzgar de otra forma: actualmente, los vendedores de sistemas de bases de datos como Oracle están entre las mayores compañías software en el mundo, y los sistemas de bases de datos forman una parte importante de la línea de productos de compañías más diversificadas, como Microsoft e IBM.

1.2. SISTEMAS DE BASES DE DATOS FRENTE A SISTEMAS DE ARCHIVOS

Considérese parte de una empresa de cajas de ahorros que mantiene información acerca de todos los clientes y cuentas de ahorros. Una manera de mantener la información en un computador es almacenarla en archivos del sistema operativo. Para permitir a los usuarios manipular la información, el sistema tiene un número de programas de aplicación que manipula los archivos, incluyendo:

- Un programa para efectuar cargos o abonos en una cuenta.
- Un programa para añadir una cuenta nueva.
- Un programa para calcular el saldo de una cuenta.
- Un programa para generar las operaciones mensuales.

Estos programas de aplicación se han escrito por programadores de sistemas en respuesta a las necesidades de la organización bancaria.

Si las necesidades se incrementan, se añaden nuevos programas de aplicación al sistema. Por ejemplo, supóngase que las regulaciones de un nuevo gobierno permiten a las cajas de ahorros ofrecer cuentas corrientes. Como resultado se crean nuevos archivos permanentes que contengan información acerca de todas las cuentas corrientes mantenidas por el banco, y puede ser necesario escribir nuevos programas de aplicación para tratar situaciones que no existían en las cuentas de ahorro, tales como manejar descubiertos. Así, sobre la marcha, se añaden más archivos y programas de aplicación al sistema.

Este **sistema de procesamiento de archivos** típico que se acaba de describir se mantiene mediante un sistema operativo convencional. Los registros permanentes son almacenados en varios archivos y se escriben diferentes programas de aplicación para extraer registros y para añadir registros a los archivos adecuados. Antes de la llegada de los sistemas de gestión de bases de datos (SGBDs), las organizaciones normalmente han almacenado la información usando tales sistemas.

Mantener información de la organización en un sistema de procesamiento de archivos tiene una serie de inconvenientes importantes:

- **Redundancia e inconsistencia de datos.** Debido a que los archivos y programas de aplicación son creados por diferentes programadores en un largo período de tiempo, los diversos archivos tienen probablemente diferentes formatos y los programas pueden estar escritos en diferentes lenguajes. Más aún, la misma información puede estar duplicada en diferentes lugares (archivos). Por ejemplo, la dirección y número de teléfono de un cliente particular puede aparecer en un archivo que contenga registros de cuentas de ahorros y en un archivo que contenga registros de una cuenta corriente. Esta redundancia conduce a un almacenamiento y coste de acceso más altos. Además, puede conducir a **inconsistencia de datos;** es decir, las diversas copias de los mismos datos pueden no coincidir. Por ejemplo, un cambio en la dirección del cliente puede estar reflejado en los registros de las cuentas de ahorro pero no estarlo en el resto del sistema.
- **Dificultad en el acceso a los datos.** Supóngase que uno de los empleados del banco necesita averiguar los nombres de todos los clientes que viven en el distrito postal 28733 de la ciudad. El empleado pide al departamento de procesamiento de datos que genere dicha lista. Debido a que esta petición no fue prevista cuando el sistema original fue diseñado, no hay un programa de aplicación a mano para satisfacerla. Hay, sin embargo, un programa de aplicación que genera la lista de *todos* los clientes. El empleado del banco tiene ahora dos opciones: bien obtener la lista de todos los clientes y obtener la información que necesita manualmente, o bien pedir al departamento de procesamiento de datos que haga

que un programador de sistemas escriba el programa de aplicación necesario. Ambas alternativas son obviamente insatisfactorias. Supóngase que se escribe tal programa y que, varios días más tarde, el mismo empleado necesita arreglar esa lista para incluir sólo aquellos clientes que tienen una cuenta con saldo de 10.000 € o más. Como se puede esperar, un programa para generar tal lista no existe. De nuevo, el empleado tiene que elegir entre dos opciones, ninguna de las cuales es satisfactoria.

La cuestión aquí es que el entorno de procesamiento de archivos convencional no permite que los datos necesarios sean obtenidos de una forma práctica y eficiente. Se deben desarrollar sistemas de recuperación de datos más interesantes para un uso general.

- **Aislamiento de datos.** Debido a que los datos están dispersos en varios archivos, y los archivos pueden estar en diferentes formatos, es difícil escribir nuevos programas de aplicación para recuperar los datos apropiados.
- **Problemas de integridad.** Los valores de los datos almacenados en la base de datos deben satisfacer ciertos tipos de **restricciones de consistencia.** Por ejemplo, el saldo de una cuenta bancaria no puede nunca ser más bajo de una cantidad predeterminada (por ejemplo 25 €). Los desarrolladores hacen cumplir esas restricciones en el sistema añadiendo el código apropiado en los diversos programas de aplicación. Sin embargo, cuando se añaden nuevas restricciones, es difícil cambiar los programas para hacer que se cumplan. El problema es complicado cuando las restricciones implican diferentes elementos de datos de diferentes archivos.
- **Problemas de atomicidad.** Un sistema de un computador, como cualquier otro dispositivo mecánico o eléctrico, está sujeto a fallo. En muchas aplicaciones es crucial asegurar que, una vez que un fallo ha ocurrido y se ha detectado, los datos se restauran al estado de consistencia que existía antes del fallo. Consideremos un programa para transferir 50 € desde la cuenta A a la B. Si ocurre un fallo del sistema durante la ejecución del programa, es posible que los 50 € fueron eliminados de la cuenta A pero no abonados a la cuenta B, resultando un estado de la base de datos inconsistente. Claramente, es esencial para la consistencia de la base de datos que ambos, el abono y el cargo tengan lugar, o que ninguno tenga lugar. Es decir, la transferencia de fondos debe ser *atómica*: ésta debe ocurrir en ellos por completo o no ocurrir en absoluto. Es difícil asegurar esta propiedad en un sistema de procesamiento de archivos convencional.
- **Anomalías en el acceso concurrente.** Conforme se ha ido mejorando el conjunto de ejecución de los sistemas y ha sido posible una respuesta en tiempo más rápida, muchos sistemas han ido permitiendo a múltiples usuarios actualizar los datos simultáneamente. En tales sistemas un entorno de interacción de actualizaciones concurrentes puede dar lugar a datos inconsistentes. Considérese una cuenta bancaria A, que contiene 500 €. Si dos clientes retiran fondos (por ejemplo 50 € y 100 € respectivamente) de la cuenta A en aproximadamente el mismo tiempo, el resultado de las ejecuciones concurrentes puede dejar la cuenta en un estado incorrecto (o inconsistente). Supongamos que los programas se ejecutan para cada retirada y escriben el resultado después. Si los dos programas funcionan concurrentemente, pueden leer ambos el valor 500 €, y escribir después 450 € y 400 €, respectivamente. Dependiendo de cuál escriba el último valor, la cuenta puede contener bien 450 € o bien 400 €, en lugar del valor correcto, 350 €. Para protegerse contra esta posibilidad, el sistema debe mantener alguna forma de supervisión. Sin embargo, ya que se puede acceder a los datos desde muchos programas de aplicación diferentes que no han sido previamente coordinados, la supervisión es difícil de proporcionar.
- **Problemas de seguridad.** No todos los usuarios de un sistema de bases de datos deberían poder acceder a todos los datos. Por ejemplo, en un sistema bancario, el personal de nóminas necesita ver sólo esa parte de la base de datos que tiene información acerca de varios empleados del banco. No necesitan acceder a la información acerca de las cuentas de clientes. Como los programas de aplicación se añaden al sistema de una forma ad hoc, es difícil garantizar tales restricciones de seguridad.

Estas dificultades, entre otras, han motivado el desarrollo de los sistemas de bases de datos. En este libro se verán los conceptos y algoritmos que han sido incluidos en los sistemas de bases de datos para resolver los problemas mencionados anteriormente. En la mayor parte de este libro se usa una empresa bancaria como el ejemplo de una aplicación corriente de procesamiento de datos típica encontrada en una empresa.

1.3. VISIÓN DE LOS DATOS

Un sistema de bases de datos es una colección de archivos interrelacionados y un conjunto de programas que permitan a los usuarios acceder y modificar estos archivos. Uno de los propósitos principales de un sistema de bases de datos es proporcionar a los usuarios una visión *abstracta* de los datos. Es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos.

1.3.1. Abstracción de datos

Para que el sistema sea útil debe recuperar los datos eficientemente. Esta preocupación ha conducido al diseño de estructuras de datos complejas para la representación de los datos en la base de datos. Como muchos usuarios de sistemas de bases de datos no están familiarizados con computadores, los desarrolladores esconden la complejidad a los usuarios a través de varios niveles de abstracción para simplificar la interacción de los usuarios con el sistema:

- **Nivel físico:** El nivel más bajo de abstracción describe *cómo* se almacenan realmente los datos. En el nivel físico se describen en detalle las estructuras de datos complejas de bajo nivel.
- **Nivel lógico:** El siguiente nivel más alto de abstracción describe *qué* datos se almacenan en la base de datos y qué relaciones existen entre esos datos. La base de datos completa se describe así en términos de un número pequeño de estructuras relativamente simples. Aunque la implementación de estructuras simples en el nivel lógico puede involucrar estructuras complejas del nivel físico, los usuarios del nivel lógico no necesitan preocuparse de esta complejidad. Los administradores de bases de datos, que deben decidir la información que se mantiene en la base de datos, usan el nivel lógico de abstracción.
- **Nivel de vistas:** El nivel más alto de abstracción describe sólo parte de la base de datos completa. A pesar del uso de estructuras más simples en el nivel lógico, queda algo de complejidad, debido a la variedad de información almacenada en una gran base de datos. Muchos usuarios del sistema de base de datos no necesitan toda esta información. En su lugar, tales usuarios necesitan acceder sólo a una parte de la base de datos. Para que su interacción con el sistema se simplifique, se define la abstracción del nivel de vistas. El sistema puede proporcionar muchas vistas para la misma base de datos.

La Figura 1.1 muestra la relación entre los tres niveles de abstracción.

Una analogía con el concepto de tipos de datos en lenguajes de programación puede clarificar la distinción entre los niveles de abstracción. La mayoría de lenguajes de programación de alto nivel soportan la estructura de tipo registro. Por ejemplo, en un lenguaje tipo Pascal, se pueden declarar registros como sigue:

```
type cliente = record
        nombre-cliente : string;
        id-cliente : string;
        calle-cliente : string;
        ciudad-cliente : string;
    end;
```

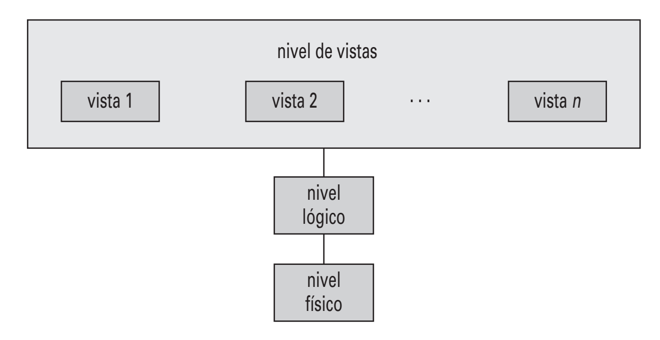


FIGURA 1.1. Los tres niveles de abstracción de datos.

Este código define un nuevo registro llamado *cliente* con cuatro campos. Cada campo tiene un nombre y un tipo asociado a él. Una empresa bancaria puede tener varios tipos de registros, incluyendo

- *cuenta*, con campos *número-cuenta* y *saldo*
- *empleado*, con campos *nombre-empleado* y *sueldo*

En el nivel físico, un registro *cliente*, *cuenta* o *empleado* se puede describir como un bloque de posiciones almacenadas consecutivamente (por ejemplo, palabras o bytes). El compilador del lenguaje esconde este nivel de detalle a los programadores. Análogamente, el sistema de base de datos esconde muchos de los detalles de almacenamiento de nivel inferior a los programadores de bases de datos. Los administradores de bases de datos pueden ser conscientes de ciertos detalles de la organización física de los datos.

En el nivel lógico cada registro de este tipo se describe mediante una definición de tipo, como se ha ilustrado en el fragmento de código previo, y se define la relación entre estos tipos de registros. Los programadores, cuando usan un lenguaje de programación, trabajan en este nivel de abstracción. De forma similar, los administradores de bases de datos trabajan habitualmente en este nivel de abstracción.

Finalmente, en el nivel de vistas, los usuarios de computadores ven un conjunto de programas de aplicación que esconden los detalles de los tipos de datos. Análogamente, en el nivel de vistas se definen varias vistas de una base de datos y los usuarios de la misma ven única y exclusivamente esas vistas. Además de esconder detalles del nivel lógico de la base de datos, las vistas también proporcionan un mecanismo de seguridad para evitar que los usuarios accedan a ciertas partes de la base de datos. Por ejemplo, los cajeros de un banco ven únicamente la parte de la base de datos que tiene información de cuentas de clientes; no pueden acceder a la información referente a los sueldos de los empleados.

1.3.2. Ejemplares y esquemas

Las bases de datos van cambiando a lo largo del tiempo conforme la información se inserta y borra. La colección de información almacenada en la base de datos en

un momento particular se denomina un *ejemplar* de la base de datos. El diseño completo de la base de datos se llama el *esquema* de la base de datos. Los esquemas son raramente modificados, si es que lo son alguna vez.

El concepto de esquemas y ejemplares de bases de datos se puede entender por analogía con un programa escrito en un lenguaje de programación. Un *esquema* de base de datos corresponde a las declaraciones de variables (junto con definiciones de tipos asociadas) en un programa. Cada variable tiene un valor particular en un instante de tiempo. Los valores de las variables en un programa en un instante de tiempo corresponde a un *ejemplar* de un esquema de bases de datos.

Los sistemas de bases de datos tiene varios esquemas divididos de acuerdo a los niveles de abstracción que se han discutido. El **esquema físico** describe el diseño físico en el nivel físico, mientras que el **esquema lógico** describe el diseño de la base de datos en el nivel lógico. Una base de datos puede tener también varios esquemas en el nivel de vistas, a menudo denominados **subesquemas**, que describen diferentes vistas de la base de datos.

De éstos, el esquema lógico es con mucho el más importante, en términos de su efecto en los programas de aplicación, ya que los programadores construyen las aplicaciones usando el esquema lógico. El esquema físico está oculto bajo el esquema lógico, y puede ser fácilmente cambiado usualmente sin afectar a los programas de aplicación. Los programas de aplicación se dice que muestran independencia física de datos si no dependen del esquema físico y, por tanto, no deben ser modificados si cambia el esquema físico.

Se estudiarán los lenguajes para la descripción de los esquemas, después de introducir la noción de modelos de datos en el siguiente apartado.

1.4. MODELOS DE LOS DATOS

Bajo la estructura de la base de datos se encuentra el **modelo de datos**: una colección de herramientas conceptuales para describir los datos, las relaciones, la semántica y las restricciones de consistencia. Para ilustrar el concepto de un modelo de datos, describimos dos modelos de datos en este apartado: el modelo entidad-relación y el modelo relacional. Los diferentes modelos de datos que se han propuesto se clasifican en tres grupos diferentes: modelos lógicos basados en objetos, modelos lógicos basados en registros y modelos físicos.

1.4.1. Modelo entidad-relación

El modelo de datos entidad-relación (E-R) está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados *entidades*, y de *relaciones* entre estos objetos. Una entidad es una «cosa» u «objeto» en el mundo real que es distinguible de otros objetos. Por ejemplo, cada persona es una entidad, y las cuentas bancarias pueden ser consideradas entidades.

Las entidades se describen en una base de datos mediante un conjunto de **atributos**. Por ejemplo, los atributos *número-cuenta* y *saldo* describen una cuenta particular de un banco y pueden ser atributos del conjunto de entidades *cuenta*. Análogamente, los atributos *nombre-cliente*, *calle-cliente* y *ciudad-cliente* pueden describir una entidad *cliente*.

Un atributo extra, *id-cliente*, se usa para identificar unívocamente a los clientes (dado que puede ser posible que haya dos clientes con el mismo nombre, dirección y ciudad. Se debe asignar un identificador único de cliente a cada cliente. En los Estados Unidos, muchas empresas utilizan el número de la seguridad social de una persona (un número único que el Gobierno de los Estados Unidos asigna a cada persona en los Estados Unidos) como identificador de cliente*.

Una **relación** es una asociación entre varias entidades. Por ejemplo, una relación *impositor* asocia un cliente con cada cuenta que tiene. El conjunto de todas las entidades del mismo tipo, y el conjunto de todas las relaciones del mismo tipo, se denominan respectivamente **conjunto de entidades** y **conjunto de relaciones**.

La estructura lógica general de una base de datos se puede expresar gráficamente mediante un *diagrama* E-R, que consta de los siguientes componentes:

- **Rectángulos**, que representan conjuntos de entidades.
- **Elipses**, que representan atributos.
- **Rombos**, que representan relaciones entre conjuntos de entidades.
- **Líneas**, que unen los atributos con los conjuntos de entidades y los conjuntos de entidades con las relaciones.

Cada componente se etiqueta con la entidad o relación que representa.

Como ilustración, considérese parte de una base de datos de un sistema bancario consistente en clientes y cuentas que tienen esos clientes. En la Figura 1.2 se

* N. del T. En España, muchas empresas usan el D.N.I. como identificador unívoco, pero a veces encuentran problemas con los números de D.N.I. que por desgracia aparecen repetidos. Para resolverlo, o bien se usa otro identificador propio de la empresa o se añade un código al número de D.N.I.

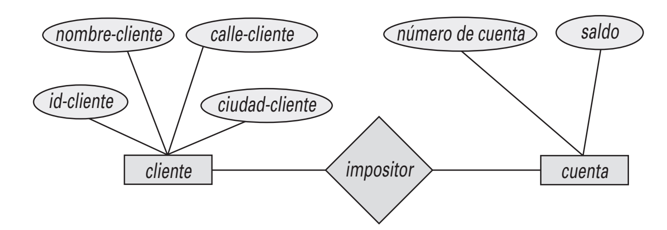


FIGURA 1.2. Ejemplo de diagrama E-R.

muestra el diagrama E-R correspondiente. El diagrama E-R indica que hay dos conjuntos de entidades *cliente* y *cuenta*, con los atributos descritos anteriormente. El diagrama también muestra la relación *impositor* entre cliente y cuenta.

Además de entidades y relaciones, el modelo E-R representa ciertas restricciones que los contenidos de la base de datos deben cumplir. Una restricción importante es la *correspondencia de cardinalidades*, que expresa el número de entidades con las que otra entidad se puede asociar a través de un conjunto de relaciones. Por ejemplo, si cada cuenta puede pertenecer sólo a un cliente, el modelo puede expresar esta restricción.

El modelo entidad-relación se utiliza habitualmente en el proceso de diseño de bases de datos, y se estudiará en produndidad en el Capítulo 2.

1.4.2. Modelo relacional

En el modelo relacional se utiliza un grupo de tablas para representar los datos y las relaciones entre ellos. Cada tabla está compuesta por varias columnas, y cada columna tiene un nombre único. En la Figura 1.3 se presenta un ejemplo de base de datos relacional consistente en tres tablas: la primera muestra los clientes de un banco, la segunda, las cuentas, y la tercera, las cuentas que pertenecen a cada cliente.

id-cliente	*nombre-cliente*	*calle-cliente*	*ciudad-cliente*
19.283.746	González	Arenal	La Granja
01.928.374	Gómez	Carretas	Cerceda
67.789.901	López	Mayor	Peguerinos
18.273.609	Abril	Preciados	Valsaín
32.112.312	Santos	Mayor	Peguerinos
33.666.999	Rupérez	Ramblas	León
01.928.374	Gómez	Carretas	Cerceda

(a) La tabla *cliente*

número-cuenta	*saldo*
C-101	500
C-215	700
C-102	400
C-305	350
C-201	900
C-217	750
C-222	700

(b) La tabla *cuenta*

id-cliente	*número-cuenta*
19.283.746	C-101
19.283.746	C-201
01.928.374	C-215
67.789.901	C-102
18.273.609	C-305
32.112.312	C-217
33.666.999	C-222
01.928.374	C-201

(b) La tabla *impositor*

FIGURA 1.3. Ejemplo de base de datos relacional.

La primera tabla, la tabla *cliente*, muestra, por ejemplo, que el cliente cuyo identificador es 19.283.746 se llama González y vive en la calle Arenal sita en La Granja. La segunda tabla, *cuenta*, muestra que las cuentas C-101 tienen un saldo de 500 € y la C-201 un saldo de 900 € respectivamente.

La tercera tabla muestra las cuentas que pertenecen a cada cliente. Por ejemplo, la cuenta C-101 pertenece al cliente cuyo identificador es 19.283.746 (González), y los clientes 19.283.746 (González) y 01.928.374 (Gómez) comparten el número de cuenta A-201 (pueden compartir un negocio).

El modelo relacional es un ejemplo de un modelo basado en registros. Los modelos basados en registros se denominan así porque la base de datos se estructura en registros de formato fijo de varios tipos. Cada tabla contiene registros de un tipo particular. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla corresponden a los atributos del tipo de registro.

No es difícil ver cómo se pueden almacenar las tablas en archivos. Por ejemplo, un carácter especial (como una coma) se puede usar para delimitar los diferentes atributos de un registro, y otro carácter especial (como un carácter de nueva línea) se puede usar para delimitar registros. El modelo relacional oculta tales detalles de implementación de bajo nivel a los desarrolladores de bases de datos y usuarios.

El modelo de datos relacional es el modelo de datos más ampliamente usado, y una amplia mayoría de sistemas de bases de datos actuales se basan en el modelo relacional. Los Capítulos 3 a 7 tratan el modelo relacional en detalle.

El modelo relacional se encuentra a un nivel de abstracción inferior al modelo de datos E-R. Los diseños de bases de datos a menudo se realizan en el modelo E-R, y después se traducen al modelo relacional; el Capítulo 2 describe el proceso de traducción. Por ejemplo, es fácil ver que las tablas *cliente* y *cuenta* corresponden a los conjuntos de entidades del mismo nombre, mientras que la tabla *impositor* corresponde al conjunto de relaciones *impositor*.

Nótese también que es posible crear esquemas en el modelo relacional que tengan problemas tales como información duplicada innecesariamente. Por ejemplo, supongamos que se almacena *número-cuenta* como un atributo del registro *cliente*. Entonces, para representar el hecho de que las cuentas C-101 y C-201 pertenecen ambas al cliente González (con identificador de cliente 19.283.746) sería necesario almacenar dos filas en la tabla *cliente*. Los valores de *nombre-cliente*, *calle-cliente* y *ciudad-cliente* de González estarían innecesariamente duplicados en las dos filas. En el Capítulo 7 se estudiará cómo distinguir buenos diseños de esquema de malos diseños.

1.4.3. Otros modelos de datos

El **modelo de datos orientado a objetos** es otro modelo de datos que está recibiendo una atención creciente.

El modelo orientado a objetos se puede observar como una extensión del modelo E-R con las nociones de encapsulación, métodos (funciones) e identidad de objeto. El Capítulo 8 examina el modelo de datos orientado a objetos.

El **modelo de datos relacional orientado a objetos** combina las características del modelo de datos orientado a objetos y el modelo de datos relacional. El Capítulo 9 lo examina.

Los modelos de datos semiestructurados permiten la especificación de datos donde los elementos de datos individuales del mismo tipo pueden tener diferentes conjuntos de atributos. Esto es diferente de los modelos de datos mencionados anteriormente, en los que cada elemento de datos de un tipo particular debe tener el mismo conjunto de atributos. El **lenguaje de marcas extensible (XML, eXtensible Markup Language)** se usa ampliamente para representar datos semiestructurados. El Capítulo 10 lo trata.

Históricamente, otros dos modelos de datos, el **modelo de datos de red** y el **modelo de datos jerárquico**, precedieron al modelo de datos relacional. Estos modelos estuvieron ligados fuertemente a la implementación subyacente y complicaban la tarea del modelado de datos. Como resultado se usan muy poco actualmente, excepto en el código de bases de datos antiguo que aún está en servicio en algunos lugares. Se describen en los apéndices A y B para los lectores interesados.

1.5. LENGUAJES DE BASES DE DATOS

Un sistema de bases de datos proporciona un **lenguaje de definición de datos** para especificar el esquema de la base de datos y un **lenguaje de manipulación de datos** para expresar las consultas a la base de datos y las modificaciones. En la práctica, los lenguajes de definición y manipulación de datos no son dos lenguajes separados; en su lugar simplemente forman partes de un único lenguaje de bases de datos, tal como SQL, ampliamente usado.

1.5.1. Lenguaje de definición de datos

Un esquema de base de datos se especifica mediante un conjunto de definiciones expresadas mediante un lenguaje especial llamado **lenguaje de definición de datos (LDD)**.

Por ejemplo, la siguiente instrucción en el lenguaje SQL define la tabla *cuenta*:

```
create table cuenta
    (número-cuenta char(10),
     saldo integer)
```

La ejecución de la instrucción LDD anterior crea la tabla *cuenta*. Además, actualiza un conjunto especial de tablas denominado **diccionario de datos** o **directorio de datos**.

Un diccionario de datos contiene **metadatos**, es decir, datos acerca de los datos. El esquema de una tabla es un ejemplo de metadatos. Un sistema de base de datos consulta el diccionario de datos antes de leer o modificar los datos reales.

Especificamos el almacenamiento y los métodos de acceso usados por el sistema de bases de datos por un conjunto de instrucciones en un tipo especial de LDD denominado lenguaje de **almacenamiento y definición de datos**. Estas instrucciones definen los detalles de implementación de los esquemas de base de datos, que se ocultan usualmente a los usuarios.

Los valores de datos almacenados en la base de datos deben satisfacer ciertas **restricciones de consistencia**. Por ejemplo, supóngase que el saldo de una cuenta no debe caer por debajo de 100 €. El LDD proporciona facilidades para especificar tales restricciones. Los sistemas de bases de datos comprueban estas restricciones cada vez que se actualiza la base de datos.

1.5.2. Lenguaje de manipulación de datos

La **manipulación de datos** es:

- La recuperación de información almacenada en la base de datos.
- La inserción de información nueva en la base de datos.
- El borrado de información de la base de datos.
- La modificación de información almacenada en la base de datos.

Un ***lenguaje de manipulación de datos* (LMD)** es un lenguaje que permite a los usuarios acceder o manipular los datos organizados mediante el modelo de datos apropiado. Hay dos tipos básicamente:

- **LMDs procedimentales.** Requieren que el usuario especifique *qué* datos se necesitan y *cómo* obtener esos datos.
- **LMDs declarativos** (también conocidos como LMDs **no procedimentales**). Requieren que el usuario especifique *qué* datos se necesitan *sin* especificar cómo obtener esos datos.

Los LMDs declarativos son más fáciles de aprender y usar que los LMDs procedimentales. Sin embargo, como el usuario no especifica cómo conseguir los datos, el sistema de bases de datos tiene que determinar un medio eficiente de acceder a los datos. El componente LMD del lenguaje SQL es no procedimental.

Una **consulta** es una instrucción de solicitud para recuperar información. La parte de un LMD que implica recuperación de información se llama **lenguaje de consultas**. Aunque técnicamente sea incorrecto, en la práctica se usan los términos *lenguaje de consultas* y *lenguaje de manipulación de datos* como sinónimos.

Esta consulta en el lenguaje SQL encuentra el nombre del cliente cuyo identificador de cliente es 19.283.746:

> **select** *cliente.nombre-cliente*
> **from** *cliente*
> **where** *cliente.id-cliente* = '19 283 746'

La consulta especifica que las filas *de* (from) la tabla *cliente donde* (where) el id-cliente es 19 283 746 se debe recuperar, y que se debe mostrar el atributo *nombre-cliente* de estas filas. Si se ejecutase la consulta con la tabla de la Figura 1.3, se mostraría el nombre González.

Las consultas pueden involucrar información de más de una tabla. Por ejemplo, la siguiente consulta encuentra el saldo de todas las cuentas pertenecientes al cliente cuyo identificador de cliente es 19 283 746.

> **select** *cuenta.saldo*
> **from** *impositor*, *cuenta*
> **where** *impositor.id-cliente* = '19-283-746' **and**
> *impositor.número-cuenta* = *cuenta.número-cuenta*

Si la consulta anterior se ejecutase con las tablas de la Figura 1.3, el sistema encontraría que las dos cuentas denominadas C-101 y C-201 pertenecen al cliente 19 283 746 e imprimiría los saldos de las dos cuentas, es decir, 500 y 900 €.

Hay varios lenguajes de consulta de bases de datos en uso, ya sea comercialmente o experimentalmente. Se estudiará el lenguaje de consultas más ampliamente usado, SQL, en el Capítulo 4. También se estudiarán otros lenguajes de consultas en el Capítulo 5.

Los niveles de abstracción que se discutieron en el Apartado 1.3 se aplican no solo a la definición o estructuración de datos, sino también a la manipulación de datos. En el nivel físico se deben definir algoritmos que permitan un acceso eficiente a los datos. En los niveles superiores de abstracción se enfatiza la facilidad de uso. El objetivo es proporcionar una interacción humana eficiente con el sistema. El componente procesador de consultas del sistema de bases de datos (que se estudia en los Capítulos 13 y 14) traduce las consultas LMD en secuencias de acciones en el nivel físico del sistema de bases de datos.

1.5.3. Acceso a la base de datos desde programas de aplicación

Los **programas de aplicación** son programas que se usan para interaccionar con la base de datos. Los programas de aplicación se escriben usualmente en un lenguaje *anfitrión*, tal como Cobol, C, C++ o Java. En el sistema bancario algunos ejemplos son programas que emiten los cheques de las nóminas, las cuentas de débito, las cuentas de crédito o las transferencias de fondos entre cuentas.

Para acceder a la base de datos, las instrucciones LMD necesitan ser ejecutadas desde el lenguaje anfitrión. Hay dos maneras de hacerlo:

- Proporcionando una interfaz de programas de aplicación (conjunto de procedimientos) que se pueden usar para enviar instrucciones LMD y LDD a la base de datos, y recuperar los resultados.

  El estándar de conectividad abierta de bases de datos (ODBC, Open Data Base Connectivity) definido por Microsoft para el uso con el lenguaje C es un estándar de interfaz de programas de aplicación usado comúnmente. El estándar conectividad de Java con bases de datos (JDBC, Java Data Base Connectivity) proporciona características correspondientes para el lenguaje Java.
- Extendiendo la sintaxis del lenguaje anfitrión para incorporar llamadas LMD dentro del programa del lenguaje anfitrión. Usualmente, un carácter especial precede a las llamadas LMD, y un preprocesador, denominado el **precompilador LMD**, convierte las instrucciones LMD en llamadas normales a procedimientos en el lenguaje anfitrión.

1.6. USUARIOS Y ADMINISTRADORES DE LA BASE DE DATOS

Un objetivo principal de un sistema de bases de datos es recuperar información y almacenar nueva información en la base de datos. Las personas que trabajan con una base de datos se pueden catalogar como usuarios de bases de datos o como administradores de bases de datos.

1.6.1. Usuarios de bases de datos e interfaces de usuario

Hay cuatro tipos diferentes de usuarios de un sistema de base de datos, diferenciados por la forma en que ellos esperan interactuar con el sistema. Se han diseñado diferentes tipo de interfaces de usuario para diferentes tipos de usuarios.

- **Usuarios normales.** Son usuarios no sofisticados que interactúan con el sistema mediante la invocación de alguno de los programas de aplicación permanentes que se ha escrito previamente. Por ejemplo, un cajero bancario que necesita transferir 50 € de la cuenta *A* a la cuenta *B* invoca un programa llamado *transferir*. Este programa pide al

cajero el importe de dinero a transferir, la cuenta de la que el dinero va a ser transferido y la cuenta a la que el dinero va a ser transferido.

Como otro ejemplo, considérese un usuario que desee encontrar su saldo de cuenta en World Wide Web. Tal usuario podría acceder a un formulario en el que introduce su número de cuenta. Un programa de aplicación en el servidor Web recupera entonces el saldo de la cuenta, usando el número de cuenta proporcionado, y pasa la información al usuario.

La interfaz de usuario normal para los usuarios normales es una interfaz de formularios, donde el usuario puede rellenar los campos apropiados del formulario. Los usuarios normales pueden también simplemente leer *informes* generados de la base de datos.

- **Programadores de aplicaciones.** Son profesionales informáticos que escriben programas de aplicación. Los programadores de aplicaciones pueden elegir entre muchas herramientas para desarrollar interfaces de usuario. Las herramientas de **desarrollo rápido de aplicaciones (DRA)** son herramientas que permiten al programador de aplicaciones construir formularios e informes sin escribir un programa. Hay también tipos especiales de lenguajes de programación que combinan estructuras de control imperativo (por ejemplo, para bucles for, bucles while e instrucciones if-then-else) con instrucciones del lenguaje de manipulación de datos. Estos lenguajes, llamados a veces *lenguajes de cuarta generación*, a menudo incluyen características especiales para facilitar la generación de formularios y la presentación de datos en pantalla. La mayoría de los sistemas de bases de datos comerciales incluyen un lenguaje de cuarta generación.
- **Los usuarios sofisticados** interactúan con el sistema sin programas escritos. En su lugar, ellos forman sus consultas en un lenguaje de consulta de bases de datos. Cada una de estas consultas se envía al *procesador de consultas*, cuya función es transformar instrucciones LMD a instrucciones que el gestor de almacenamiento entienda. Los analistas que envían las consultas para explorar los datos en la base de datos entran en esta categoría.

  Las herramientas de **procesamiento analítico en línea (OLAP, Online Analytical Processing)** simplifican la labor de los analistas permitiéndoles ver resúmenes de datos de formas diferentes. Por ejemplo, un analista puede ver las ventas totales por región (por ejemplo, norte, sur, este y oeste), o por producto, o por una combinación de la región y del producto (es decir, las ventas totales de cada producto en cada región). Las herramientas también permiten al analista seleccionar regiones específicas, examinar los datos con más detalle (por ejemplo, ventas por ciudad dentro de una región) o examinar los datos con menos detalle (por ejemplo, agrupando productos por categoría).

  Otra clase de herramientas para los analistas son las herramientas de **recopilación de datos**, que les ayudan a encontrar ciertas clases de patrones de datos.

  En el Capítulo 22 se estudiarán las herramientas de recopilación de datos.
- **Usuarios especializados.** Son usuarios sofisticados que escriben aplicaciones de bases de datos especializadas que no son adecuadas en el marco de procesamiento de datos tradicional. Entre estas aplicaciones están los sistemas de diseño asistido por computador, sistemas de bases de conocimientos y sistemas expertos, sistemas que almacenan los datos con tipos de datos complejos (por ejemplo, datos gráficos y datos de audio) y sistemas de modelado del entorno. Varias de estas aplicaciones se tratan en los Capítulos 8 y 9.

1.6.2. Administrador de la base de datos

Una de las principales razones de usar SGBDs es tener un control centralizado tanto de los datos como de los programas que acceden a esos datos. La persona que tiene este control central sobre el sistema se llama **administrador de la base de datos (ABD)**. Las funciones del ABD incluyen las siguientes:

- **Definición del esquema.** El ABD crea el esquema original de la base de datos escribiendo un conjunto de instrucciones de definición de datos en el LDD.
- **Definición de la estructura y del método de acceso.**
- **Modificación del esquema y de la organización física.** Los ABD realizan cambios en el esquema y en la organización física para reflejar las necesidades cambiantes de la organización, o para alterar la organización física para mejorar el rendimiento.
- **Concesión de autorización para el acceso a los datos.** La concesión de diferentes tipos de autorización permite al administrador de la base de datos determinar a qué partes de la base de datos puede acceder cada usuario. La información de autorización se mantiene en una estructura del sistema especial que el sistema de base de datos consulta cuando se intenta el acceso a los datos en el sistema.
- **Mantenimiento rutinario.** Algunos ejemplos de actividades rutinarias de mantenimiento del administrado de la base de datos son:
  - Copia de seguridad periódica de la base de datos, bien sobre cinta o sobre servidores remotos, para prevenir la pérdida de datos en caso de desastres como inundaciones.

— Asegurarse de que haya suficiente espacio libre en disco para las operaciones normales y aumentar el espacio en disco según sea necesario.

— Supervisión de los trabajos que se ejecuten en la base de datos y asegurarse de que el rendimiento no se degrada por tareas muy costosas iniciadas por algunos usuarios.

1.7. GESTIÓN DE TRANSACCIONES

Varias operaciones sobre la base de datos forman a menudo una única unidad lógica de trabajo. Un ejemplo que se vio en el Apartado 1.2 es la transferencia de fondos, en el que una cuenta (*A*) se carga y otra cuenta (*B*) se abona. Claramente es esencial que, o bien tanto el cargo como el abono tengan lugar, o bien no ocurra ninguno. Es decir, la transferencia de fondos debe ocurrir por completo o no ocurrir en absoluto. Este requisito de todo o nada se denomina **atomicidad**. Además, es esencial que la ejecución de la transferencia de fondos preserve la consistencia de la base de datos. Es decir, el valor de la suma $A + B$ se debe preservar. Este requisito de corrección se llama **consistencia**. Finalmente, tras la ejecución correcta de la transferencia de fondos, los nuevos valores de las cuentas *A* y *B* deben persistir, a pesar de la posibilidad de fallo del sistema. Este requisito de persistencia se llama **durabilidad**.

Una **transacción** es una colección de operaciones que se lleva a cabo como una única función lógica en una aplicación de bases de datos. Cada transacción es una unidad de atomicidad y consistencia. Así, se requiere que las transacciones no violen ninguna restricción de consistencia de la base de datos. Es decir, si la base de datos era consistente cuando la transacción comenzó, la base de datos debe ser consistente cuando la transacción termine con éxito. Sin embargo, durante la ejecución de una transacción, puede ser necesario permitir inconsistencias temporalmente, ya que o el cargo de *A* o el abono de *B* se debe realizar uno antes que otro. Esta inconsistencia temporal, aunque necesaria, puede conducir a dificultades si ocurre un fallo.

Es responsabilidad del programador definir adecuadamente las diferentes transacciones, de tal manera que cada una preserve la consistencia de la base de datos. Por ejemplo, la transacción para transferir fondos de la cuenta *A* a la cuenta *B* se podría definir como compuesta de dos programas separados: uno que carga la cuenta A y otro que abona la cuenta B. La ejecución de estos dos programas uno después del otro preservará realmente la consistencia. Sin embargo, cada programa en sí mismo no transforma la base de datos de un estado consistente en otro nuevo estado consistente. Así, estos programas no son transacciones.

Asegurar las propiedades de atomicidad y durabilidad es responsabilidad del propio sistema de bases de datos, específicamente del **componente de gestión de transacciones**. En ausencia de fallos, toda transacción completada con éxito y atómica se archiva fácilmente. Sin embargo, debido a diversos tipos de fallos, una transacción puede no siempre completar su ejecución con éxito. Si se asegura la propiedad de atomicidad, una transacción que falle no debe tener efecto en el estado de la base de datos. Así, la base de datos se restaura al estado en que estaba antes de que la transacción en cuestión comenzara su ejecución. El sistema de bases de datos debe realizar la **recuperación de fallos**, es decir, detectar los fallos del sistema y restaurar la base de datos al estado que existía antes de que ocurriera el fallo.

Finalmente, cuando varias transacciones actualizan la base de datos concurrentemente, la consistencia de los datos puede no ser preservada, incluso aunque cada transacción individualmente sea correcta. Es responsabilidad del **gestor de control de concurrencia** controlar la interacción entre las transacciones concurrentes para asegurar la consistencia de la base de datos.

Los sistemas de bases de datos diseñados para uso sobre pequeños computadores personales pueden no tener todas las características vistas. Por ejemplo, muchos sistemas pequeños imponen la restricción de permitir el acceso a un único usuario a la base de datos en un instante de tiempo. Otros dejan las tareas de copias de seguridad y recuperación a los usuarios. Estas restricciones permiten un gestor de datos más pequeño, con menos requisitos de recursos físicos, especialmente de memoria principal. Aunque tales enfoques de bajo coste y prestaciones son suficientes para bases de datos personales pequeñas, son inadecuadas para satisfacer las necesidades de una empresa de media a gran escala.

1.8. ESTRUCTURA DE UN SISTEMA DE BASES DE DATOS

Un sistema de bases de datos se divide en módulos que se encargan de cada una de las responsabilidades del sistema completo. Los componentes funcionales de un sistema de bases de datos se pueden dividir a grandes rasgos en los componentes gestor de almacenamiento y procesador de consultas.

El gestor de consultas es importante porque las bases de datos requieren normalmente una gran cantidad de

espacio de almacenamiento. Las bases de datos corporativas tienen un tamaño de entre cientos de gigabytes y, para las mayores bases de datos, terabytes de datos. Un gigabyte son 1.000 megabytes (1.000 millones de bytes), y un terabyte es 1 millón de megabytes (1 billón de bytes). Debido a que la memoria principal de los computadores no puede almacenar esta gran cantidad de información, esta se almacena en discos. Los datos se trasladan entre el disco de almacenamiento y la memoria principal cuando es necesario. Como la transferencia de datos a y desde el disco es lenta comparada con la velocidad de la unidad central de procesamiento, es fundamental que el sistema de base de datos estructure los datos para minimizar la necesidad de movimiento de datos entre el disco y la memoria principal.

El procesador de consultas es importante porque ayuda al sistema de bases de datos a simplificar y facilitar el acceso a los datos. Las vistas de alto nivel ayudan a conseguir este objetivo. Con ellas, los usuarios del sistema no deberían ser molestados innecesariamente con los detalles físicos de implementación del sistema. Sin embargo, el rápido procesamiento de las actualizaciones y de las consultas es importante. Es trabajo del sistema de bases de datos traducir las actualizaciones y las consultas escritas en un lenguaje no procedimental, en el nivel lógico, en una secuencia de operaciones en el nivel físico.

1.8.1. Gestor de almacenamiento

Un *gestor de almacenamiento* es un módulo de programa que proporciona la interfaz entre los datos de bajo nivel en la base de datos y los programas de aplicación y consultas emitidas al sistema. El gestor de almacenamiento es responsable de la interacción con el gestor de archivos. Los datos en bruto se almacenan en disco usando un sistema de archivos, que está disponible habitualmente en un sistema operativo convencional. El gestor de almacenamiento traduce las diferentes instrucciones LMD a órdenes de un sistema de archivos de bajo nivel. Así, el gestor de almacenamiento es responsable del almacenamiento, recuperación y actualización de los datos en la base de datos.

Los componentes del gestor de almacenamiento incluyen:

- **Gestor de autorización e integridad**, que comprueba que se satisfagan las restricciones de integridad y la autorización de los usuarios para acceder a los datos.
- **Gestor de transacciones**, que asegura que la base de datos quede en un estado consistente (correcto) a pesar de los fallos del sistema, y que las ejecuciones de transacciones concurrentes ocurran si conflictos.
- **Gestor de archivos**, que gestiona la reserva de espacio de almacenamiento de disco y las estructuras de datos usadas para representar la información almacenada en disco.
- **Gestor de memoria intermedia**, que es responsable de traer los datos del disco de almacenamiento a memoria principal y decidir qué datos tratar en memoria caché. El gestor de memoria intermedia es una parte crítica del sistema de bases de datos, ya que permite que la base de datos maneje tamaños de datos que son mucho mayores que el tamaño de la memoria principal.

El gestor de almacenamiento implementa varias estructuras de datos como parte de la implementación física del sistema:

- **Archivos de datos**, que almacenan la base de datos en sí.
- **Diccionario de datos**, que almacena metadatos acerca de la estructura de la base de datos, en particular, el esquema de la base de datos.
- **Índices**, que proporcionan acceso rápido a elementos de datos que tienen valores particulares.

1.8.2. Procesador de consultas

Los componentes del procesador de consultas incluyen:

- **Intérprete del LDD**, que interpreta las instrucciones del LDD y registra las definiciones en el diccionario de datos.
- **Compilador del LMD**, que traduce las instrucciones del LMD en un lenguaje de consultas a un plan de evaluación que consiste en instrucciones de bajo nivel que entiende el motor de evaluación de consultas.

  Una consulta se puede traducir habitualmente en varios planes de ejecución alternativos que proporcionan el mismo resultado. El compilador del LMD también realiza **optimización de consultas**, es decir, elige el plan de evaluación de menor coste de entre todas las alternativas.
- **Motor de evaluación de consultas**, que ejecuta las instrucciones de bajo nivel generadas por el compilador del LMD.

En la Figura 1.4 se muestran estos componentes y sus conexiones.

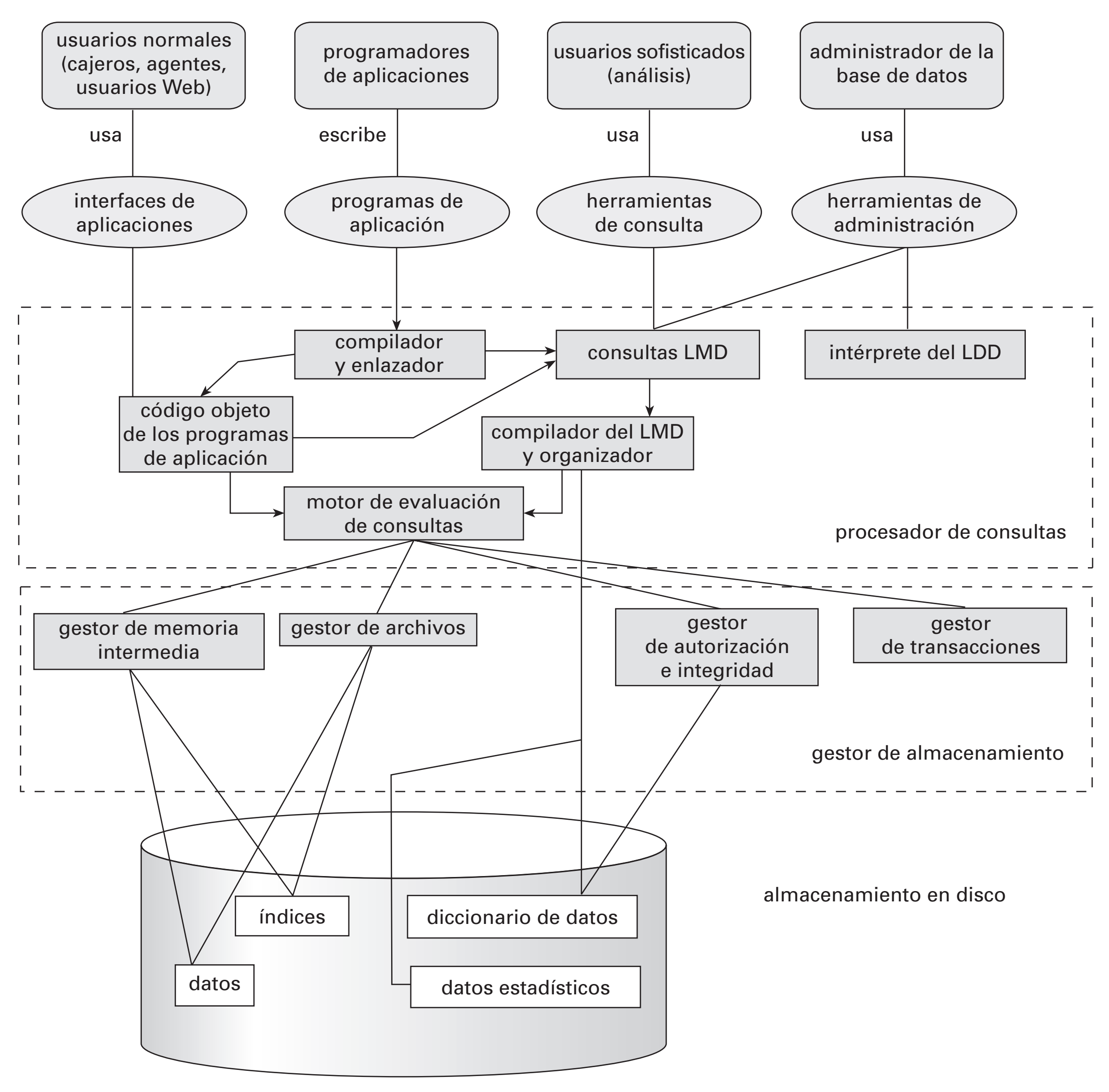


FIGURA 1.4. Estructura del sistema.

1.9. ARQUITECTURAS DE APLICACIONES

La mayoría de usuarios de un sistema de bases de datos no están situados actualmente junto al sistema de bases de datos, sino que se conectan a él a través de una red. Se puede diferenciar entonces entre las máquinas **cliente**, en donde trabajan los usuarios remotos de la base de datos, y las máquinas **servidor**, en las que se ejecuta el sistema de bases de datos.

Las aplicaciones de bases de datos se dividen usualmente en dos o tres partes, como se ilustra en la Figura 1.5. En una **arquitectura de dos capas**, la aplicación se divide en un componente que reside en la máquina cliente, que llama a la funcionalidad del sistema de bases de datos en la máquina servidor mediante instrucciones del lenguaje de consultas. Los estándares de interfaces de programas de aplicación como ODBC y JDBC se usan para la interacción entre el cliente y el servidor.

En cambio, en una **arquitectura de tres capas**, la máquina cliente actúa simplemente como frontal y no contiene ninguna llamada directa a la base de datos. En su lugar, el cliente se comunica con un **servidor de aplicaciones**, usualmente mediante una interfaz de formularios. El servidor de aplicaciones, a su vez, se comunica con el sistema de bases de datos para acceder a los datos. La **lógica de negocio** de la aplicación, que establece las acciones a realizar bajo determinadas condiciones, se incorpora en el servidor de aplicaciones, en lugar de ser distribuida a múltiples clientes. Las aplicaciones de tres capas son más apropiadas para grandes aplicaciones, y para las aplicaciones que se ejecutan en World Wide Web.

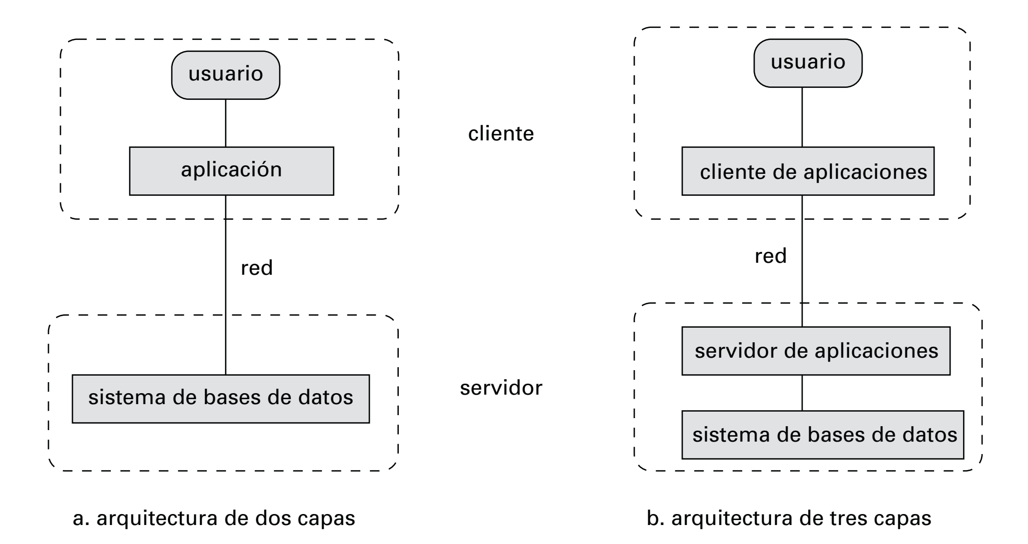


FIGURA 1.5. Arquitecturas de dos y tres capas.

1.10. HISTORIA DE LOS SISTEMAS DE BASES DE DATOS

El procesamiento de datos impulsa el crecimiento de los computadores, como ocurriera en los primeros días de los computadores comerciales. De hecho, la automatización de las tareas de procesamiento de datos precede a los computadores. Las tarjetas perforadas, inventadas por Hollerith, se usaron en los principios del siglo xx para registrar los datos del censo de los EE.UU., y se usaron sistemas mecánicos para procesar las tarjetas y para tabular los resultados. Las tarjetas perforadas posteriormente se usaron ampliamente como medio para introducir datos en los computadores.

Las técnicas del almacenamiento de datos han evolucionado a lo largo de los años:

- **Década de 1950 y principios de la década de 1960.** Se desarrollaron las cintas magnéticas para el almacenamiento de datos. Las tareas de procesamiento de datos tales como las nóminas fueron automatizadas, con los datos almacenados en cintas. El procesamiento de datos consistía en leer datos de una o más cintas y escribir datos en una nueva cinta. Los datos también se podían introducir desde paquetes de tarjetas perforadas e impresos en impresoras. Por ejemplo, los aumentos de sueldo se procesaban introduciendo los aumentos en las tarjetas perforadas y leyendo el paquete de cintas perforadas en sincronización con una cinta que contenía los detalles maestros de los salarios. Los registros debían estar igualmente ordenados. Los aumentos de sueldo tenían que añadirse a los sueldos leídos de la cinta maestra, y escribirse en una nueva cinta; esta nueva cinta se convertía en la nueva cinta maestra.

  Las cintas (y los paquetes de tarjetas perforadas) sólo se podían leer secuencialmente, y los tamaños de datos eran mucho mayores que la memoria principal; así, los programas de procesamiento de datos tenían que procesar los datos según un determinado orden, leyendo y mezclando datos de cintas y paquetes de tarjetas perforadas.

- **Finales de la década de 1960 y la década de 1970.** El amplio uso de los discos fijos a finales de la década de 1960 cambió en gran medida el escenario del procesamiento de datos, ya que los discos fijos permitieron el acceso directo a los datos. La ubicación de los datos en disco no era importante, ya que a cualquier posición del disco se podía acceder en sólo decenas de milisegundo. Los datos se liberaron de la tiranía de la secuencialidad. Con los discos pudieron desarrollarse las bases de datos de red y jerárquicas, que permitieron que las estructuras de datos tales como listas y árboles pudieran almacenarse en disco. Los programadores pudieron construir y manipular estas estructuras de datos.

  Un artículo histórico de Codd [1970] definió el modelo relacional y formas no procedimentales de consultar los datos en el modelo relacional, y nacieron las bases de datos relacionales. La simplicidad del modelo relacional y la posibilidad de ocultar completamente los detalles de implementación al programador fueron realmente atractivas. Codd obtuvo posteriormente el prestigioso premio Turing de la ACM (Association of Computing Machinery, asociación de maquinaria informática) por su trabajo.

- **Década de 1980.** Aunque académicamente interesante, el modelo relacional no se usó inicialmente en la práctica debido a sus inconvenientes por el rendimiento; las bases de datos relacionales no pudieron competir con el rendimiento de las bases

de datos de red y jerárquicas existentes. Esta situación cambió con System R, un proyecto innovador en IBM Research que desarrolló técnicas para la construcción de un sistema de bases de datos relacionales eficiente. En Astrahan et al. [1976] y Chamberlin et al. [1981] se pueden encontrar excelentes visiones generales de System R. El prototipo de System R completamente funcional condujo al primer producto de bases de datos relacionales de IBM: SQL/DS. Los primeros sistemas de bases de datos relacionales, como DB2 de IBM, Oracle, Ingres y Rdb de DEC, jugaron un importante papel en el desarrollo de técnicas para el procesamiento eficiente de consultas declarativas. En los principios de la década de 1980 las bases de datos relacionales llegaron a competir con los sistemas de bases de datos jerárquicas y de red incluso en el área de rendimiento. Las bases de datos relacionales fueron tan sencillas de usar que finalmente reemplazaron a las bases de datos jerárquicas y de red; los programadores que usaban estas bases de datos estaban forzados a tratar muchos detalles de implementación de bajo nivel y tenían que codificar sus consultas de forma procedimental. Aún más importante, debían tener presente el rendimiento durante el diseño de sus programas, lo que implicaba un gran esfuerzo. En cambio, en una base de datos relacional, casi todas estas tareas de bajo nivel se realizan automáticamente por la base de datos, liberando al programador en el nivel lógico. Desde su escalada en el dominio en la década de 1980, el modelo relacional ha conseguido el reinado supremo entre todos los modelos de datos.

La década de 1980 también fue testigo de una gran investigación en las bases de datos paralelas y distribuidas, así como del trabajo inicial en las bases de datos orientadas a objetos.

- **Principios de la década de 1990.** El lenguaje SQL se diseñó fundamentalmente para las aplicaciones de ayuda a la toma de decisiones, que son intensivas en consultas, mientras que el objetivo principal de las bases de datos en la década de 1980 fue las aplicaciones de procesamiento de transacciones, que son intensivas en actualizaciones. La ayuda a la toma de decisiones y las consultas reemergieron como una importante área de aplicación para las bases de datos. Las herramientas para analizar grandes cantidades de datos experimentaron un gran crecimiento de uso.

  Muchos vendedores de bases de datos introdujeron productos de bases de datos paralelas en este periodo, así como también comenzaron ofrecer bases de datos relacionales orientadas a objeto.

- **Finales de la década de 1990.** El principal acontecimiento fue el crecimiento explosivo de World Wide Web. Las bases de datos se implantaron mucho más extensivamente que nunca antes. Los sistemas de bases de datos tienen ahora soporte para tasas de transacciones muy altas, así como muy alta fiabilidad y disponibilidad 24×7 (disponibilidad 24 horas al día y 7 días a la semana, que significa que no hay tiempos de inactividad debidos a actividades de mantenimiento planificadas). Los sistemas de bases de datos también tuvieron interfaces Web a los datos.

1.11. RESUMEN

- Un **sistema gestor de bases de datos** (SGBD) consiste en una colección de datos interrelacionados y una colección de programas para acceder a esos datos. Los datos describen una empresa particular.
- El objetivo principal de un SGBD es proporcionar un entorno que sea tanto conveniente como eficiente para las personas que lo usan para la recuperación y almacenamiento de la información.
- Los sistemas de bases de datos se diseñan para almacenar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para el almacenamiento de la información como la provisión de mecanismos para la manipulación de la información. Además, los sistemas de bases de datos deben proporcionar la seguridad de la información almacenada, en caso de caídas del sistema o intentos de accesos sin autorización. Si los datos están compartidos por varios usuarios, el sistema debe evitar posibles resultados anómalos.
- Un propósito principal de un sistema de bases de datos es proporcionar a los usuarios una visión abstracta de los datos. Es decir, el sistema esconde ciertos detalles de cómo los datos se almacenan y mantienen.
- Por debajo de la estructura de la base de datos está el **modelo de datos**: una colección de herramientas conceptuales para describir los datos, las relaciones entre los datos, la semántica de los datos y las restricciones de los datos. El modelo de datos entidad-relación es un modelo de datos ampliamente usado, y proporciona una representación gráfica conveniente para ver los datos, las relaciones y las restricciones. El modelo de datos relacional se usa ampliamente para almacenar datos en las bases de datos. Otros modelos de datos son el modelo de datos orientado a objetos, el relacional orientado a objetos y modelos de datos semiestructurados.
- El diseño general de la base de datos se denomina el **esquema** de la base de datos. Un esquema de base de datos se especifica con un conjunto de definiciones

que se expresan usando un **lenguaje de definición de datos (LDD).**

- Un **lenguaje de manipulación de datos (LMD)** es un lenguaje que permite a los usuarios acceder o manipular los datos. Los LMD no procedimentales, que requieren que un usuario especifique sólo los datos que necesita, se usan ampliamente hoy día.
- Los usuarios de bases de datos se pueden catalogar en varias clases, y cada clase de usuario usa habitualmente diferentes tipos de interfaces de la base de datos.
- Un sistema de bases de datos tiene varios subsistemas:
  — El subsistema gestor de transacciones es el responsable de asegurar que la base de datos permanezca en un estado consistente (correcto) a pesar de los fallos del sistema. El gestor de transacciones también asegura que las ejecuciones de transacciones concurrentes ocurran sin conflictos.
  — El subsistema procesador de consultas compila y ejecuta instrucciones LDD y LMD.
  — El subsistema gestor de almacenamiento es un módulo de programa que proporciona la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y las consultas enviadas al sistema.
- Las aplicaciones de bases de datos se dividen normalmente en un parte frontal que se ejecuta en las máquinas cliente y una parte que se ejecuta en el dorsal. En las arquitecturas de dos capas, el frontal se comunica directamente con una base de datos que se ejecuta en el dorsal. En las arquitecturas de tres capas, la parte dorsal se divide asimismo en un servidor de aplicaciones y en un servidor de bases de datos.

TÉRMINOS DE REPASO

- Abstracción de datos.
- Administrador de la base de datos (ADB).
- Aplicaciones de sistemas de bases de datos.
- Concurrencia.
- Diccionario de datos.
- Ejemplar de la base de datos.
- Esquema.
  — Esquema de la base de datos.
  — Esquema físico.
  — Esquema lógico.
- Inconsistencia de datos.
- Independencia física de los datos.
- Lenguajes de bases de datos.
  — Lenguaje de consultas.
  — Lenguaje de definición de datos.
- Lenguaje de manipulación de datos.
- Máquinas cliente y servidor.
- Metadatos.
- Modelos de datos.
  — Modelo de datos orientado a objetos.
  — Modelo de datos relacional.
  — Modelo de datos relacional orientado a objetos.
  — Modelo entidad-relación.
- Programa de aplicación.
- Restricciones de consistencia.
- Sistema de gestión de bases de datos (SGBD).
- Sistemas de archivos.
- Transacciones.
- Vistas de datos.

EJERCICIOS

1.1. ¿Cuáles son las cuatro diferencias principales entre un sistema de procesamiento de archivos y un SGBD?

1.2. En este capítulo se han descrito las diferentes ventajas principales de un sistema gestor de bases de datos. ¿Cuáles son los dos inconvenientes?

1.3. Explíquese la diferencia entre independencia de datos física y lógica.

1.4. Lístense las cinco responsabilidades del sistema gestor de la base de datos. Para cada responsabilidad explíquense los problemas que ocurrirían si no se realizara esa función.

1.5. ¿Cuáles son las cinco funciones principales del administrador de la base de datos?

1.6. Lístense siete lenguajes de programación que sean procedimentales y dos que sean no procedimentales. ¿Qué grupo es más fácil de aprender a usar? Explíquese la respuesta.

1.7. Lístense los seis pasos principales que se deberían dar en la realización de una base de datos para una empresa particular.

1.8. Considérese un *array* de enteros bidimensional de tamaño $n \times m$ que se va a usar en su lenguaje de programación preferido. Usando el *array* como ejemplo, ilústrese la diferencia (a) entre los tres niveles de abstracción y (b) entre esquema y ejemplares.

NOTAS BIBLIOGRÁFICAS

A continuación se listan libros de propósito general, colecciones de artículos de investigación y sitios Web de bases de datos.

Libros de texto que tratan los sistemas de bases de datos incluyen Abiteboul et al. [1995], Date [1995], Elmasri y Navathe [2000], O'Neil y O'Neil [2000], Ramakrishnan y Gehrke [2000] y Ullman [1988]. El tratamiento del procesamiento de transacciones en libros de texto se puede encontrar en Bernstein y Newcomer [1997] y Gray y Reuter [1993].

Varios libros incluyen colecciones de artículos de investigación sobre la gestión de bases de datos. Entre estos están Bancilhon y Buneman [1990], Date [1986], Date [1990], Kim [1995], Zaniolo et al. [1997], y Stonebraker y Hellerstein [1998].

Una revisión de los logros en la gestión de bases de datos y una valoración de los desafíos en la investigación futura aparece en Silberschatz et al. [1990], Silberschatz et al. [1996] y Bernstein et al. [1998]. La página inicial del grupo especial de interés de la ACM en gestión de datos (véase www.acm.org/sigmod) proporciona una gran cantidad de información sobre la investigación en bases de datos. Los sitios Web de los vendedores de bases de datos (véase el apartado Herramientas a continuación) proporciona detalles acerca de sus respectivos productos.

Codd [1970] es el artículo histórico que introdujo el modelo relacional. En Fry y Sibley [1976] y Sibley [1976] se ofrecen discusiones referentes a la evolución de los SGBDs y al desarrollo de la tecnología de bases de datos.

HERRAMIENTAS

Hay un gran número de sistemas de bases de datos comerciales en uso actualmente. Los principales incluyen: DB2 de IBM (www.ibm.com/software/data), Oracle (www.oracle.com), Microsoft SQL Server (www.microsoft.com/sql), Informix (www.informix.com) y Sybase (www.sybase.com). Algunos de estos sistemas están disponibles gratuitamente para uso personal o no comercial, o para desarrollo, pero no para implantación real.

Hay también una serie de sistemas de bases de datos gratuitos/públicos; algunos ampliamente usados incluyen MySQL (www.mysql.com) y PostgresSQL (www.postgresql.org).

Una lista más completa de enlaces a vendedores y otra información se encuentra disponible en la página inicial de este libro en www.research.bell-labs.com/topic/books/db-book.

PARTE

I

MODELOS DE DATOS

Un **modelo de datos** es una colección de herramientas conceptuales para la descripción de datos, relaciones entre datos, semántica de los datos y restricciones de consistencia. En esta parte se estudiarán dos modelos de datos —el modelo entidad-relación y el modelo relacional.

El modelo entidad-relación (E-R) es un modelo de datos de alto nivel. Está basado en una percepción de un mundo real que consiste en una colección de objetos básicos, denominados *entidades*, y de *relaciones* entre estos objetos.

El modelo relaciona es un modelo de menor nivel. Usa una colección de tablas para representar tanto los datos como las relaciones entre los datos. Su simplicidad conceptual ha conducido a su adopción general; actualmente, una vasta mayoría de productos de bases de datos se basan en el modelo relacional. Los diseñadores formulan generalmente el diseño del esquema de la base de datos modelando primero los datos en alto nivel, usando el modelo E-R, y después traduciéndolo al modelo relacional.

Se estudiarán otros modelos de datos más tarde en este libro. El modelo de datos orientado a objetos, por ejemplo, extiende la representación de entidades añadiendo nociones de encapsulación, métodos (funciones) e identidad de objeto. El modelo de datos relacional orientado a objetos combina características del modelo de datos orientado a objetos y del modelo de datos relacional. Los Capítulos 8 y 9 tratan respectivamente estos dos modelos de datos.

CAPÍTULO

2 MODELO ENTIDAD-RELACIÓN

EL modelo de datos **entidad-relación (E-R)** está basado en una percepción del mundo real consistente en objetos básicos llamados *entidades* y de *relaciones* entre estos objetos. Se desarrolló para facilitar el diseño de bases de datos permitiendo la especificación de un *esquema de la empresa* que representa la estructura lógica completa de una base de datos. El modelo de datos E-R es uno de los diferentes modelos de datos semánticos; el aspecto semántico del modelo yace en la representación del significado de los datos. El modelo E-R es extremadamente útil para hacer corresponder los significados e interacciones de las empresas del mundo real con un esquema conceptual. Debido a esta utilidad, muchas herramientas de diseño de bases de datos se basan en los conceptos del modelo E-R.

2.1. CONCEPTOS BÁSICOS

Hay tres nociones básicas que emplea el modelo de datos E-R: conjuntos de entidades, conjuntos de relaciones y atributos.

2.1.1. Conjuntos de entidades

Una **entidad** es una «cosa» u «objeto» en el mundo real que es distinguible de todos los demás objetos. Por ejemplo, cada persona en un desarrollo es una entidad. Una entidad tiene un conjunto de propiedades, y los valores para algún conjunto de propiedades pueden identificar una entidad de forma unívoca. Por ejemplo, el D.N.I. 67.789.901 identifica unívocamente una persona particular en la empresa. Análogamente, se puede pensar en los préstamos bancarios como entidades, y un número de préstamo P-15 en la sucursal de Castellana identifica unívocamente una entidad de préstamo. Una entidad puede ser concreta, como una persona o un libro, o puede ser abstracta, como un préstamo, unas vacaciones o un concepto.

Un **conjunto de entidades** es un conjunto de entidades del mismo tipo que comparten las mismas propiedades, o atributos. El conjunto de todas las personas que son clientes en un banco dado, por ejemplo, se pueden definir como el conjunto de entidades *cliente*. Análogamente, el conjunto de entidades *préstamo* podría representar el conjunto de todos los préstamos concedidos por un banco particular. Las entidades individuales que constituyen un conjunto se llaman la *extensión* del conjunto de entidades. Así, todos los clientes de un banco son la extensión del conjunto de entidades *cliente*.

Los conjuntos de entidades no son necesariamente disjuntos. Por ejemplo, es posible definir el conjunto de entidades de todos los empleados de un banco (*empleado*) y el conjunto de entidades de todos los clientes del banco (*cliente*). Una entidad *persona* puede ser una entidad *empleado*, una entidad *cliente*, ambas cosas, o ninguna.

Una entidad se representa mediante un conjunto de **atributos**. Los atributos describen propiedades que posee cada miembro de un conjunto de entidades. La designación de un atributo para un conjunto de entidades expresa que la base de datos almacena información similar concerniente a cada entidad del conjunto de entidades; sin embargo, cada entidad puede tener su propio valor para cada atributo. Posibles atributos del conjunto de entidades *cliente* son *id-cliente*, *nombre-cliente, calle-cliente* y *ciudad-cliente*. En la vida real, habría más atributos, tales como el número de la calle, el número del portal, la provincia, el código postal, y la comunidad autónoma, pero no se incluyen en el ejemplo simple. Posibles atributos del conjunto de entidades *préstamo* son *número-préstamo* e *importe*.

Cada entidad tiene un **valor** para cada uno de sus atributos. Por ejemplo, una entidad *cliente* en concreto puede tener el valor 32.112.312 para *id-cliente*, el valor Santos para *nombre-cliente*, el valor Mayor para *calle-cliente* y el valor Peguerinos para *ciudad-cliente*.

El atributo *id-cliente* se usa para identificar unívocamente a los clientes, dado que no hay más de un cliente con el mismo nombre, calle y ciudad. En los Estados Unidos, muchas empresas encuentran conveniente usar el número *seguridad-social* de una persona[1] como un

[1] En España se asigna a cada persona del país un número único, denominado número del documento nacional de identidad (D.N.I.) para identificarla unívocamente. Se supone que cada persona tiene un único D.N.I., y no hay dos personas con el mismo D.N.I.

atributo cuyo valor identifica unívocamente a la persona. En general la empresa tendría que crear y asignar un identificador a cada cliente.

Para cada atributo hay un conjunto de valores permitidos, llamados el **dominio**, o el **conjunto de valores**, de ese atributo. El dominio del atributo *nombre-cliente* podría ser el conjunto de todas las cadenas de texto de una cierta longitud. Análogamente, el dominio del atributo *número-préstamo* podría ser el conjunto de todas las cadenas de la forma «P-*n*», donde *n* es un entero positivo.

Una base de datos incluye así una colección de conjuntos de entidades, cada una de las cuales contiene un número de entidades del mismo tipo. En la Figura 2.1 se muestra parte de una base de datos de un banco que consta de dos conjuntos de entidades, *cliente* y *préstamo*.

Formalmente, un atributo de un conjunto de entidades es una función que asigna al conjunto de entidades un dominio. Como un conjunto de entidades puede tener diferentes atributos, cada entidad se puede describir como un conjunto de pares (atributo,valor), un par para cada atributo del conjunto de entidades. Por ejemplo, una entidad concreta *cliente* se puede describir mediante el conjunto {(*id-cliente*, 67.789.901), (*nombre-cliente*, López), (*calle-cliente*, Mayor), (*ciudad-cliente*, Peguerinos)}, queriendo decir que la entidad describe una persona llamada López que tiene D.N.I. número 67.789.901, y reside en la calle Mayor en Peguerinos. Se puede ver, en este punto, que existe una integración del esquema abstracto con el desarrollo real de la empresa que se está modelando. Los valores de los atributos que describen una entidad constituirán una porción significante de los datos almacenados en la base de datos.

Un atributo, como se usa en el modelo E-R, se puede caracterizar por los siguientes tipos de atributo.

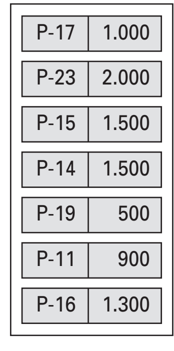
- Atributos **simples** y **compuestos**. En los ejemplos considerados hasta ahora, los atributos han sido simples; es decir, no están divididos en subpartes. Los atributos compuestos, en cambio, se pueden dividir en subpartes (es decir, en otros atributos). Por ejemplo, *nombre-cliente* podría estar estructurado como un atributo compuesto consistente en *nombre*, *primer-apellido* y *segundo-apellido*. Usar atributos compuestos en un esquema de diseño es una buena elección si el usuario desea referirse a un atributo completo en algunas ocasiones y, en otras, a algún componente del atributo. Se podrían haber sustituido los atributos del conjunto de entidades *cliente*, *calle-cliente* y *ciudad-cliente*, por el atributo compuesto *dirección-cliente*, con los atributos *calle, ciudad, provincia,* y *código-postal*[2]. Los atributos compuestos ayudan a agrupar los atributos relacionados, haciendo los modelos más claros.

  Nótese también que un atributo compuesto puede aparecer como una jerarquía. Volviendo al ejemplo del atributo compuesto *dirección-cliente*, su componente *calle* puede ser a su vez dividido en *número-calle*, *nombre-calle* y *piso*. Estos ejemplos de atributos compuestos para el conjunto de entidades *cliente* se representa en la Figura 2.2.
- Atributos **monovalorados** y **multivalorados**. Los atributos que se han especificado en los ejemplos tienen todos un valor sólo para una entidad concreta. Por ejemplo, el atributo *número-préstamo* para una entidad préstamo específico, referencia a un único número de préstamo. Tales atributos se llaman **monovalorados**. Puede haber ocasiones en las que un atributo tiene un conjunto de valores para una entidad específica. Considérese un conjunto de entidades *empleado* con el atributo *número-teléfono*. Cualquier empleado particular puede tener cero, uno o más números de teléfono. Este tipo de atributo se llama *multivalorado*. En ellos, se pueden colocar apropiadamente límites inferior y superior en el número de valores en el atributo **multivalorado**. Como otro ejemplo, un atributo *nombre-subordinado* del conjunto de entidades *empleado*

[2] Se asume el formato de *calle-cliente* y *dirección* usado en España, que incluye un código postal numérico llamado «código postal».

Santos	32.112.312	Mayor	Peguerinos
Gómez	01.928.374	Carretas	Cerceda
López	67.789.901	Mayor	Peguerinos
Sotoca	55.555.555	Real	Cádiz
Pérez	24.466.880	Carretas	Cerceda
Valdivieso	96.396.396	Goya	Vigo
Fernández	33.557.799	Jazmín	León

cliente



P-17	1.000
P-23	2.000
P-15	1.500
P-14	1.500
P-19	500
P-11	900
P-16	1.300

préstamo

FIGURA 2.1. Conjunto de entidades *cliente* y *préstamo*.

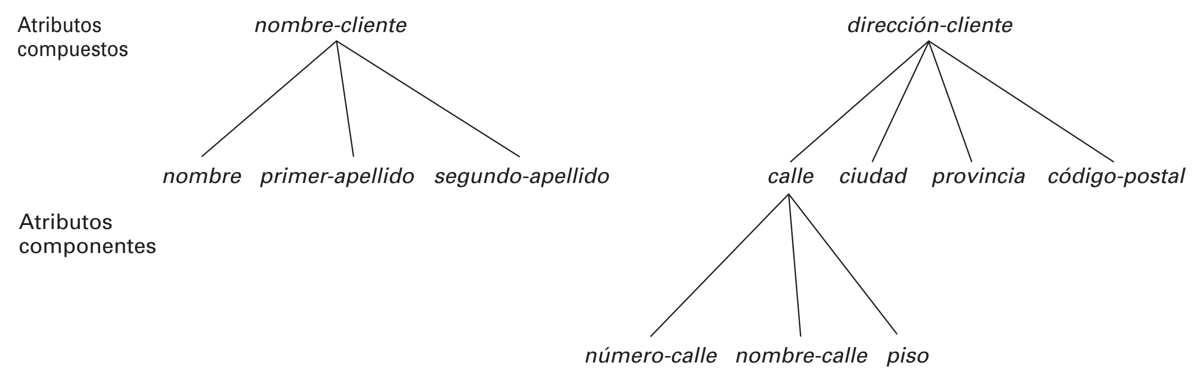


FIGURA 2.2. Atributos compuestos *nombre-cliente* y *dirección-cliente*.

sería multivalorado, ya que un empleado en concreto podría tener cero, uno o más subordinados.

Cuando sea apropiado se pueden establecer límites superior e inferior en el número de valores de un atributo multivalorado. Por ejemplo, un banco puede limitar el número de números de teléfono almacenados para un único cliente a dos. Colocando límites en este caso, se expresa que el atributo *número-teléfono* del conjunto de entidades *cliente* puede tener entre cero y dos valores.

- Atributos **derivados**. El valor para este tipo de atributo se puede derivar de los valores de otros atributos o entidades relacionados. Por ejemplo, sea el conjunto de entidades *cliente* que tiene un atributo *préstamos* que representa cuántos préstamos tiene un cliente en el banco. Ese atributo se puede derivar contando el número de entidades *préstamo* asociadas con ese *cliente*.

  Como otro ejemplo, considérese que el conjunto de entidades *empleado* tiene un atributo *edad*, que indica la edad del cliente. Si el conjunto de entidades *cliente* tiene también un atributo *fecha-de-nacimiento*, se puede calcular *edad* a partir de *fecha-de-nacimiento* y de la fecha actual. Así, *edad* es un atributo derivado. En este caso, *fecha-de-nacimiento* y *antigüedad* pueden serlo, ya que representan el primer día en que el empleado comenzó a trabajar para el banco y el tiempo total que el empleado lleva trabajando para el banco, respectivamente. El valor de *antigüedad* se puede derivar del valor de *fecha-comienzo* y de la fecha actual. En este caso, *fecha-comienzo* se puede conocer como atributo *base* o atributo *almacenado*. El valor de un atributo derivado no se almacena, sino que se calcula cuando sea necesario.

Un atributo toma un valor **nulo** cuando una entidad no tiene un valor para un atributo. El valor *nulo* también puede indicar «no aplicable», es decir, que el valor no existe para la entidad. Por ejemplo, una persona puede no tener segundo nombre de pila. *Nulo* puede también designar que el valor de un atributo es desconocido. Un valor desconocido puede ser, bien *perdido* (el valor existe pero no se tiene esa información) o *desconocido* (no se conoce si el valor existe realmente o no).

Por ejemplo, si el valor *nombre* para un *cliente* particular es *nulo*, se asume que el valor es perdido, ya que cada cliente debe tener un nombre. Un valor *nulo* para el atributo *piso* podría significar que la dirección no incluye un piso (no aplicable), que existe piso pero no se conoce cuál es (perdido), o que no se sabe si el piso forma parte o no de la dirección del cliente (desconocido).

Una base de datos para una empresa bancaria puede incluir diferentes conjuntos de entidades. Por ejemplo, además del mantenimiento de clientes y préstamos, el banco también proporciona cuentas, que se representan mediante el conjunto de entidades *cuenta* con atributos *número-cuenta* y *saldo*. También, si el banco tiene un número de sucursales diferentes, se puede mantener información acerca de todas las sucursales del banco. Cada conjunto de entidades *sucursal* se describe mediante los atributos *nombre-sucursal*, *ciudad-sucursal* y *activo*.

2.1.2. Conjuntos de relaciones

Una **relación** es una asociación entre diferentes entidades. Por ejemplo, se puede definir una relación que asocie al cliente López con el préstamo P-15. Esta relación especifica que López es un cliente con el préstamo número P-15.

Un **conjunto de relaciones** es un conjunto de relaciones del mismo tipo. Formalmente es una relación matemática con $n >= 2$ de conjuntos de entidades (posiblemente no distintos). Si $E_1, E_2, \ldots, E_n$ son conjuntos de entidades, entonces un conjunto de relaciones R es un subconjunto de:

$$\{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

donde $(e_1, e_2, \ldots e_n)$ es una relación.

Considérense las dos entidades *cliente* y *préstamo* de la Figura 2.1. Se define el conjunto de relaciones *prestatario* para denotar la asociación entre clientes y préstamos bancarios que los clientes tengan. Esta asociación se describe en la Figura 2.3.

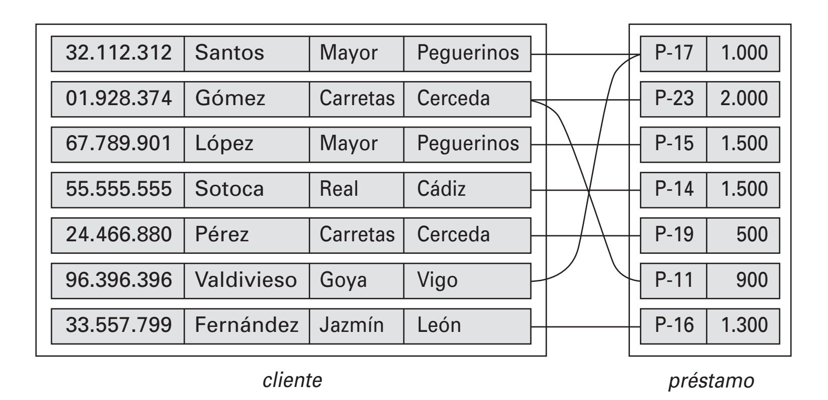


FIGURA 2.3. Conjunto de relaciones *prestatario*.

Como otro ejemplo, considérense los dos conjuntos de entidades *préstamo* y *sucursal*. Se puede definir el conjunto de relaciones *sucursal-préstamo* para denotar la asociación entre un préstamo y la sucursal en que se mantiene ese préstamo.

La asociación entre conjuntos de entidades se conoce como *participación*; es decir, los conjuntos de entidades E_1, E_2,…, E_n **participan** en el conjunto de relaciones R. Un **ejemplar de relación** en un esquema E-R representa que existe una asociación entre las entidades denominadas en la empresa del mundo real que se modela. Como ilustración, el *cliente* individual López, que tiene D.N.I. 67.789.901, y la entidad *préstamo* P-15 participan en un ejemplar de relación de *prestatario*. Este ejemplar de relación representa que, en la empresa del mundo real, la persona llamada López cuyo número de D.N.I. es 67.789.901 ha tomado un préstamo que está numerado como P-15.

La función que desempeña una entidad en una relación se llama **papel** de la entidad.

Debido a que los conjuntos de entidades que participan en un conjunto de relaciones son generalmente distintos, los papeles están implícitos y no se especifican normalmente. Sin embargo, son útiles cuando el significado de una relación necesita aclaración. Tal es el caso cuando los conjuntos de entidades de una relación no son distintos; es decir, el mismo conjunto de entidades participa en una relación más de una vez con diferentes papeles. En este tipo de conjunto de relaciones, que se llama algunas veces conjunto de relaciones **recursivo**, es necesario hacer explícitos los papeles para especificar cómo participa una entidad en un ejemplar de relación. Por ejemplo, considérese una conjunto de entidades *empleado* que almacena información acerca de todos los empleados del banco. Se puede tener un conjunto de relaciones *trabaja-para* que se modela mediante pares ordenados de entidades *empleado*. El primer empleado de un par toma el papel de *trabajador*, mientras el segundo toma el papel de *jefe*. De esta manera, todas las relaciones *trabaja-para* son pares (trabajador, jefe); los pares (jefe, trabajador) están excluidos.

Una relación puede también tener **atributos descriptivos**. Considérese un conjunto de relaciones *impositor* con conjuntos de entidades *cliente* y *cuenta*. Se podría asociar el atributo *fecha-acceso* a esta relación para especificar la fecha más reciente en que un cliente accedió a una cuenta. La relación *impositor* entre las entidades correspondientes al cliente García y la cuenta C-217 se describen mediante {(*fecha-acceso*, 23 mayo 2002)}, lo que significa que la última vez que García accedió a la cuenta C-217 fue el 23 de mayo de 2002.

Como otro ejemplo de atributos descriptivos para relaciones, supóngase que se tienen los conjuntos de entidades *estudiante* y *asignatura* que participan en una relación *matriculado*. Se podría desear almacenar un atributo descriptivo para *créditos* con la relación, para registrar si el estudiante se ha matriculado de la asignatura para obtener créditos o sólo como oyente.

Un ejemplar de relación en un conjunto de relaciones determinado debe ser identificado unívocamente a partir de sus entidades participantes, sin usar los atributos descriptivos. Para comprender este punto supóngase que deseemos modelar todas las fechas en las que un cliente ha accedido a una cuenta. El atributo monovalorado *fecha-acceso* puede almacenar sólo una única fecha de acceso. No se pueden representar varias fechas de acceso por varios ejemplares de relación entre el mismo cliente y cuenta, ya que los ejemplares de relación no estarían identificados unívocamente por las entidades participantes. La forma correcta de manejar este caso es crear un atributo multivalorado *fechas-acceso* que pueda almacenar todas las fechas de acceso.

Sin embargo, puede haber más de un conjunto de relaciones que involucren los mismos conjuntos de entidades. En nuestro ejemplo los conjuntos de entidades *cliente* y *préstamo* participan en el conjunto de relaciones *prestatario*. Además, supóngase que cada préstamo deba tener otro cliente que sirva como avalista para el préstamo. Entonces los conjuntos de entidades *cliente* y *préstamo* pueden participar en otro conjunto de relaciones: *avalista*.

Los conjuntos de relaciones *prestatario* y *sucursal-préstamo* proporcionan un ejemplo de un conjunto de relaciones **binario**, es decir, uno que implica dos conjuntos de entidades. La mayoría de los conjuntos de relaciones en un sistema de bases de datos son binarios.

Ocasionalmente, sin embargo, los conjuntos de relaciones implican más de dos conjuntos de entidades.

Por ejemplo, considérense los conjuntos de entidades *empleado*, *sucursal* y *trabajo*. Ejemplos de las entidades *trabajo* podrían ser director, cajero, auditor y otros. Las entidades trabajo pueden tener los atributos *puesto* y *nivel*. El conjunto de relaciones *trabaja-en* entre *empleado*, *sucursal* y *trabajo* es un ejemplo de una relación ternaria. Una relación ternaria entre Santos, Navacerrada y director indica que Santos actúa de director de la sucursal Navacerrada. Santos también podría actuar como auditor de la sucursal Centro, que estaría representado por otra relación. Podría haber otra relación entre Gómez, Centro y cajero, indicando que Gómez actúa como cajero en la sucursal Centro.

El número de conjuntos de entidades que participan en un conjunto de relaciones es también el **grado** del conjunto de relaciones. Un conjunto de relaciones binario tiene grado 2; un conjunto de relaciones ternario tiene grado 3.

2.2. RESTRICCIONES

Un esquema de desarrollo E-R puede definir ciertas restricciones a las que los contenidos de la base de datos se deben adaptar. En este apartado se examina la correspondencia de cardinalidades y las restricciones de participación, que son dos de los tipos más importantes de restricciones.

2.2.1. Correspondencia de cardinalidades

La **correspondencia de cardinalidades**, o razón de cardinalidad, expresa el número de entidades a las que otra entidad puede estar asociada vía un conjunto de relaciones.

La correspondencia de cardinalidades es la más útil describiendo conjuntos de relaciones binarias, aunque ocasionalmente contribuye a la descripción de conjuntos de relaciones que implican más de dos conjuntos de entidades. Este apartado se centrará en conjuntos de relaciones binarias únicamente.

Para un conjunto de relaciones binarias *R* entre los conjuntos de entidades *A* y *B*, la correspondencia de cardinalidades debe ser una de las siguientes:

- **Uno a uno**. Una entidad en *A* se asocia con *a lo sumo* una entidad en *B*, y una entidad en *B* se asocia con *a lo sumo* una entidad en *A* (véase la Figura 2.4a).

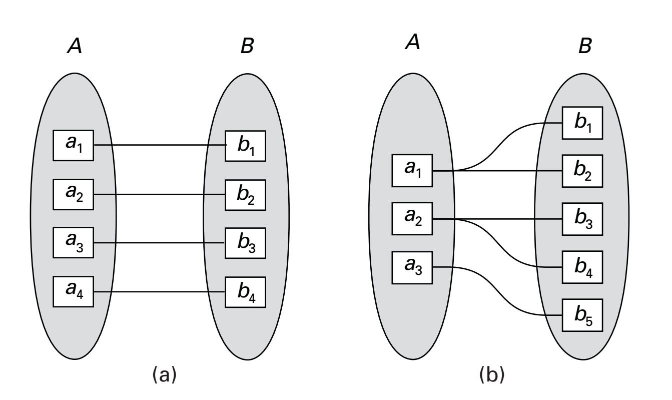


FIGURA 2.4. Correspondencia de cardinalidades. (a) Uno a uno. (b) Uno a varios.

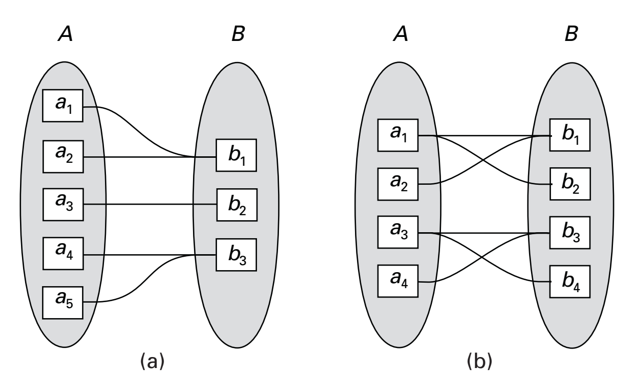


FIGURA 2.5. Correspondencia de cardinalidades. (a) Varios a uno. (b) Varios a varios.

- **Uno a varios**. Una entidad en *A* se asocia con cualquier número de entidades en *B* (ninguna o varias). Una entidad en *B*, sin embargo, se puede asociar con *a lo sumo* una entidad en *A* (véase la Figura 2.4b).
- **Varios a uno**. Una entidad en *A* se asocia con *a lo sumo* una entidad en *B*. Una entidad en *B*, sin embargo, se puede asociar con cualquier número de entidades (ninguna o varias) en *A* (véase la Figura 2.5a).
- **Varios a varios**. Una entidad en *A* se asocia con cualquier número de entidades (ninguna o varias) en *B*, y una entidad en *B* se asocia con cualquier número de entidades (ninguna o varias) en *A* (véase la Figura 2.5b).

La correspondencia de cardinalidades apropiada para un conjunto de relaciones particular depende obviamente de la situación del mundo real que el conjunto de relaciones modela.

Como ilustración considérese el conjunto de relaciones *prestatario*. Si en un banco particular un préstamo puede pertenecer únicamente a un cliente y un cliente puede tener varios préstamos, entonces el conjunto de relaciones de *cliente* a *préstamo* es uno a varios. Si un préstamo puede pertenecer a varios clientes (como préstamos que se toman en conjunto por varios socios de un negocio) el conjunto de relaciones es varios a varios. Este tipo de relación se describe en la Figura 2.3.

2.2.2. Restricciones de participación

La participación de un conjunto de entidades *E* en un conjunto de relaciones *R* se dice que es **total** si cada entidad en *E* participa al menos en una relación en R. Si sólo algunas entidades en *E* participan en relaciones en *R*, la participación del conjunto de entidades *E* en la relación *R* se llama **parcial**. Por ejemplo, se puede esperar que cada entidad *préstamo* esté relacionada con al menos un cliente mediante la relación *prestatario*. Por lo tanto, la participación de *préstamo* en el conjunto de relaciones *prestatario* es total. En cambio, un individuo puede ser cliente de un banco tenga o no tenga un préstamo en el banco. Así, es posible que sólo algunas de las entidades *cliente* estén relacionadas con el conjunto de entidades *préstamo* mediante la relación *prestatario*, y la participación de *cliente* en el conjunto de relaciones *prestatario* es por lo tanto parcial.

2.3. CLAVES

Es necesario tener una forma de especificar cómo las entidades dentro de un conjunto de entidades dado y las relaciones dentro de un conjunto de relaciones dado son distinguibles. Conceptualmente las entidades y relaciones individuales son distintas; desde una perspectiva de bases de datos, sin embargo, la diferencia entre ellas se debe expresar en término de sus atributos.

Por lo tanto, los valores de los atributos de una entidad deben ser tales que permitan *identificar unívocamente* a la entidad. En otras palabras, no se permite que ningún par de entidades tengan exactamente los mismos valores de sus atributos.

Una *clave* permite identificar un conjunto de atributos suficiente para distinguir las entidades entre sí. Las claves también ayudan a identificar unívocamente a las relaciones y así a distinguir las relaciones entre sí.

2.3.1. Conjuntos de entidades

Una **superclave** es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar de forma única una entidad en el conjunto de entidades. Por ejemplo, el atributo *id-cliente* del conjunto de entidades *cliente* es suficiente para distinguir una entidad *cliente* de las otras. Así, *id-cliente* es una superclave. Análogamente, la combinación de *nombre-cliente* e *id-cliente* es una superclave del conjunto de entidades *cliente*. El atributo *nombre-cliente* de *cliente* no es una superclave, porque varias personas podrían tener el mismo nombre.

El concepto de una superclave no es suficiente para lo que aquí se propone, ya que, como se ha visto, una superclave puede contener atributos innecesarios. Si *K* es una superclave, entonces también lo es cualquier superconjunto de *K*. A menudo interesan las superclaves tales que los subconjuntos propios de ellas no son superclave. Tales superclaves mínimas se llaman **claves candidatas**.

Es posible que conjuntos distintos de atributos pudieran servir como clave candidata. Supóngase que una combinación de *nombre-cliente* y *calle-cliente* es suficiente para distinguir entre los miembros del conjunto de entidades *cliente*. Entonces, los conjuntos {*id-cliente*} y {*nombre-cliente*, *calle-cliente*} son claves candidatas. Aunque los atributos *id-cliente* y *nombre-cliente* juntos puedan distinguir entidades *cliente*, su combinación no forma una clave candidata, ya que el atributo *id-cliente* por sí solo es una clave candidata.

Se usará el término **clave primaria** para denotar una clave candidata que es elegida por el diseñador de la base de datos como elemento principal para identificar las entidades dentro de un conjunto de entidades. Una clave (primaria, candidata y superclave) es una propiedad del conjunto de entidades, más que de las entidades individuales. Cualesquiera dos entidades individuales en el conjunto no pueden tener el mismo valor en sus atributos clave al mismo tiempo. La designación de una clave representa una restricción en el desarrollo del mundo real que se modela.

Las claves candidatas se deben designar con cuidado. Como se puede comprender, el nombre de una persona es obviamente insuficiente, ya que hay mucha gente con el mismo nombre. En España, el D.N.I. puede ser una clave candidata. Como los no residentes en España normalmente no tienen D.N.I., las empresas internacionales pueden generar sus propios identificadores únicos. Una alternativa es usar alguna combinación única de otros atributos como clave.

La clave primaria se debería elegir de manera que sus atributos nunca, o muy raramente, cambien. Por ejemplo, el campo dirección de una persona no debería formar parte de una clave primaria, porque probablemente cambiará. Los números de D.N.I., por otra parte, es seguro que no cambiarán. Los identificadores únicos generados por empresas generalmente no cambian, excepto si se fusionan dos empresas; en tal caso el mismo identificador puede haber sido emitido por ambas empresas y es necesario la reasignación de identificadores para asegurarse de que sean únicos.

2.3.2. Conjuntos de relaciones

La clave primaria de un conjunto de entidades permite distinguir entre las diferentes entidades del conjunto. Se necesita un mecanismo similar para distinguir entre las diferentes relaciones de un conjunto de relaciones.

Sea *R* un conjunto de relaciones que involucra los conjuntos de entidades $E_1, E_2, \ldots, E_n$. Sea *clave-prima-*

ria(E_i) el conjunto de atributos que forma la clave primaria para el conjunto de entidades E_i.

Asúmase por el momento que los nombres de los atributos de todas las claves primarias son únicos y que cada conjunto de entidades participa sólo una vez en la relación. La composición de la clave primaria para un conjunto de relaciones depende de la estructura de los atributos asociados al conjunto de relaciones *R*.

Si el conjunto de relaciones *R* no tiene atributos asociados, entonces el conjunto de atributos:

$$\textit{clave-primaria}(E_1) \cup \textit{clave-primaria}(E_2) \cup \ldots \cup \textit{clave-primaria}(E_n)$$

describe una relación individual en el conjunto *R*.

Si el conjunto de relaciones *R* tiene atributos $a_1, a_2, \ldots, a_m$ asociados a él, entonces el conjunto de atributos

$$\textit{clave-primaria}(E_1) \cup \textit{clave-primaria}(E_2) \cup \ldots \cup \textit{clave-primaria}(E_n) \cup \{a_1, a_2, \ldots, a_m\}$$

describe una relación individual en el conjunto *R*.

En ambos casos, el conjunto de atributos

$$\textit{clave-primaria}(E_1) \cup \textit{clave-primaria}(E_2) \cup \ldots \cup \textit{clave-primaria}(E_n)$$

forma una superclave para el conjunto de relaciones.

En el caso de que los nombres de atributos de las claves primarias no sean únicos en todos los conjuntos de entidades, los atributos se renombran para distinguirlos; el nombre del conjunto de entidades combinado con el atributo formaría un nombre único. En el caso de que un conjunto de entidades participe más de una vez en un conjunto de relaciones (como en la relación *trabaja-para* del Apartado 2.1.2) el nombre del papel se usa en lugar del nombre del conjunto de entidades para formar un nombre único de atributo.

La estructura de la clave primaria para el conjunto de relaciones depende de la correspondencia de cardinalidades asociada al conjunto de relaciones. Como ilustración, considérese el conjunto de entidades *cliente* y *cuenta*, y un conjunto de relaciones *impositor*, con el atributo *fecha-acceso* del Apartado 2.1.2. Supóngase que el conjunto de relaciones es varios a varios. Entonces la clave primaria de *impositor* consiste en la unión de las claves primarias de *cliente* y *cuenta*. Sin embargo, si un cliente puede tener sólo una cuenta —es decir, si la relación *impositor* es varios a uno de *cliente* a *cuenta*— entonces la clave primaria de *impositor* es simplemente la clave primaria de *cliente*. Análogamente, si la relación es varios a uno de *cuenta* a *cliente* —es decir, cada cuenta pertenece a lo sumo a un cliente— entonces la clave primaria de *impositor* es simplemente la clave primaria de *cuenta*. Para relaciones uno a uno se puede usar cualquier clave primaria.

Para las relaciones no binarias, si no hay restricciones de cardinalidad, entonces la superclave formada como se describió anteriormente en este apartado es la única clave candidata, y se elige como clave primaria. La elección de la clave primaria es más complicada si aparecen restricciones de cardinalidad. Ya que no se ha discutido cómo especificar restricciones de cardinalidad en relaciones no binarias, no se discutirá este aspecto en este capítulo. Se considerará este aspecto con más detalle en el apartado 7.3.

2.4. CUESTIONES DE DISEÑO

Las nociones de conjunto de entidades y conjunto de relaciones no son precisas, y es posible definir un conjunto de entidades y las relaciones entre ellas de diferentes formas. En este apartado se examinan cuestiones básicas de diseño de un esquema de bases de datos E-R. El proceso de diseño se trata con más detalle en el Apartado 2.7.4.

2.4.1. Uso de conjuntos de entidades o atributos

Considérese el conjunto de entidades *empleado* con los atributos *nombre-empleado* y *número-teléfono*. Se puede argumentar fácilmente que un *teléfono* es una entidad por sí misma con atributos *número-teléfono* y *ubicación* (la oficina donde está ubicado el teléfono). Si se toma este punto de vista, el conjunto de entidades *empleado* debe ser redefinido como sigue:

- El conjunto de entidades *empleado* con el atributo *nombre-empleado*
- El conjunto de entidades *teléfono* con atributos *número-teléfono* y *ubicación*
- La relación *empleado-teléfono*, que denota la asociación entre empleados y los teléfonos que tienen.

¿Cuál es, entonces, la diferencia principal entre esas dos definiciones de un empleado? Al tratar un teléfono como un atributo *número-teléfono* implica que cada empleado tiene precisamente un número de teléfono. Al tratar un teléfono como una entidad *teléfono* permite que los empleados puedan tener varios números de teléfono (incluido ninguno) asociados a ellos. Sin embargo, se podría definir fácilmente *número-teléfono* como un atributo multivalorado para permitir varios teléfonos por empleado.

La diferencia principal es que al tratar un teléfono como una entidad se modela mejor una situación en la que se puede querer almacenar información extra sobre un teléfono, como su ubicación, su tipo (móvil, videoteléfono o fijo) o quiénes comparten un teléfono. Así, al tratar un teléfono como una entidad es más general que tratarlo como un atributo y es apropiado cuando la generalidad pueda ser de utilidad.

En cambio, no sería adecuado tratar el atributo *nombre-empleado* como una entidad; es difícil argumentar que *nombre-empleado* sea una entidad por sí mismo (a diferencia del teléfono). Así, es apropiado tener *nombre-empleado* como un atributo del conjunto de entidades *empleado*.

Por tanto, aparecen dos cuestiones naturales: ¿qué constituye un atributo? y ¿qué constituye un conjunto de entidades? Por desgracia no hay respuestas simples. Las distinciones dependen principalmente de la estructura de la empresa del mundo real que se esté modelando y de la semántica asociada con el atributo en cuestión.

Un error común es usar la clave primaria de un conjunto de entidades como un atributo de otro conjunto de entidades, en lugar de usar una relación. Por ejemplo, es incorrecto modelar *id-cliente* como un atributo de *préstamo* incluso si cada préstamo tiene sólo un cliente. La relación *prestatario* es la forma correcta de representar la conexión entre préstamos y clientes, ya que hace su conexión explícita en lugar de implícita mediante un atributo.

Otro error relacionado que se comete es designar a los atributos de la clave primaria de los conjuntos de entidades relacionados como atributos del conjunto de relaciones. Esto no se debería hacer, ya que los atributos de la clave primaria son ya implícitos en la relación.

2.4.2. Uso de conjuntos de entidades o conjuntos de relaciones

No siempre está claro si es mejor expresar un objeto mediante un conjunto de entidades o mediante un conjunto de relaciones. En el Apartado 2.1.1 se asumió que un préstamo se modelaba como una entidad. Una alternativa es modelar un préstamo no como una entidad, sino como una relación entre clientes y sucursales, con *número-préstamo* e *importe* como atributos descriptivos. Cada préstamo se representa mediante una relación entre un cliente y una sucursal.

Si cada préstamo está asociado exactamente con un cliente y con una sucursal, se puede encontrar satisfactorio el diseño en el que un préstamo se representa como una relación. Sin embargo, con este diseño no se puede representar convenientemente una situación en que varios clientes comparten un préstamo. Habría que definir una relación separada para cada prestatario de ese préstamo común. Entonces habría que replicar los valores para los atributos descriptivos *número-préstamo* e *importe* en cada una de estas relaciones. Cada una de estas relaciones debe, por supuesto, tener el mismo valor para los atributos descriptivos *número-préstamo* e *importe*.

Surgen dos problemas como resultado de esta réplica: 1) los datos se almacenan varias veces, desperdiciando espacio de almacenamiento; y 2) las actualizaciones dejan potencialmente los datos en un estado inconsistente, en el que los valores difieren en dos relaciones para atributos que se supone tienen el mismo valor. El asunto de cómo evitar esta réplica se trata formalmente mediante la *teoría de la normalización*, discutida en el Capítulo 7.

El problema de la réplica de los atributos *número-préstamo* e *importe* no aparece en el diseño original del Apartado 2.1.1. porque *préstamo* es un conjunto de entidades.

Una posible guía para determinar si usar un conjunto de entidades o un conjunto de relaciones es designar un conjunto de relaciones para describir una acción que ocurre entre entidades. Este enfoque puede también ser útil para decidir si ciertos atributos se pueden expresar más apropiadamente como relaciones.

2.4.3. Conjuntos de relaciones binarias o *n*-arias

Las relaciones en las bases de datos son generalmente binarias. Algunas relaciones que parecen no ser binarias podrían ser representadas mejor con varias relaciones binarias. Por ejemplo, uno podría crear una relación ternaria *padres*, que relaciona un hijo con su padre y su madre. Sin embargo, tal relación se podría representar por dos relaciones binarias *padre* y *madre*, relacionando un hijo con su padre y su madre por separado. Al usar las dos relaciones *padre* y *madre* se permite registrar la madre de un niño incluso si no se conoce la identidad del padre; en la relación ternaria *padres* se necesitaría usar un valor nulo. En este caso es preferible usar conjuntos de relaciones binarias.

De hecho, siempre es posible reemplazar un conjunto de relaciones no binarias (n-aria, para $n > 2$) por un número de diferentes conjuntos de relaciones binarias. Por simplicidad, considérese el conjunto de relaciones abstracto R, ternario ($n = 3$), y los conjuntos de entidades A, B, y C. Se sustituye el conjunto de relaciones R por un conjunto de entidades E y se crean tres conjuntos de relaciones:

- R_A, relacionando E y A
- R_B, relacionando E y B
- R_C, relacionando E y C

Si el conjunto de relaciones R tiene atributos, éstos se asignan al conjunto de entidades E; por otra parte se crea un atributo de identificación especial para E (debido a que cada conjunto de entidades debe tener al menos un atributo para distinguir los miembros del conjunto).

Para cada relación (a_i,b_i,c_i) del conjunto de relaciones *R*, se crea una nueva entidad e_i en el conjunto de entidades *E*. Entonces, en cada uno de los tres nuevos conjuntos de relaciones, se inserta un nuevo miembro como sigue:

- (e_i,a_i) en R_A
- (e_i,b_i) en R_B
- (e_i,c_i) en R_C

Se puede generalizar este proceso de una forma semejante a conjuntos de relaciones *n*-arias. Así, conceptualmente, se puede restringir el modelo E-R para incluir sólo conjuntos de relaciones binarias. Sin embargo, esta restricción no siempre es deseable.

- Un atributo de identificación puede haber sido creado para el conjunto de entidades para representar el conjunto de relaciones. Este atributo, con los conjuntos de relaciones extra necesarios, incrementa la complejidad del diseño y (como se verá en el Apartado 2.9) los requisitos de almacenamiento.
- Un conjunto de relaciones *n*-arias muestra más claramente que varias entidades participan en una relación simple.
- Podría no haber una forma de traducir restricciones en la relación ternaria en restricciones sobre relaciones binarias. Por ejemplo, considérese una restricción que dice que *R* es varios a uno de *A*, *B* a *C*; es decir, cada par de entidades de *A* y *B* se asocia con a lo sumo una entidad *C*. Esta restricción no se puede expresar usando restricciones de cardinalidad sobre los conjuntos de relaciones R_A, R_B y R_C.

Considérese el conjunto de relaciones *trabaja-en* del Apartado 2.1.2 que relaciona *empleado*, *sucursal* y *trabajo*. No se puede dividir directamente *trabaja-en* en relaciones binarias entre *empleado* y *sucursal* y entre *empleado* y *trabajo*. Si se hiciese habría que registrar que Santos es director y auditor y que Santos trabaja en Navacerrada y Centro; sin embargo, no se podría registrar que Santos es director de Navacerrada y auditor de Centro, pero que no es auditor de Navacerrada y director de Centro.

El conjunto de relaciones *trabaja-en* se puede dividir en relaciones binarias creando nuevos conjuntos de entidades como se describió anteriormente. Sin embargo, no sería muy natural.

2.4.4. Ubicación de los atributos de las relaciones

La razón de cardinalidad de una relación puede afectar a la situación de los atributos de la relación. Los atributos de los conjuntos de relaciones uno a uno o uno a varios pueden estar asociados con uno de los conjuntos de entidades participantes, en lugar de con el conjunto de relaciones. Por ejemplo, especificamos que *impositor* es un conjunto de relaciones uno a varios tal que un cliente puede tener varias cuentas, pero cada cuenta está asociada únicamente con un cliente. En este caso, el atributo *fecha-acceso*, que especifica cuándo accedió por última vez el cliente a la cuenta, podría estar asociado con el conjunto de entidades *cuenta*, como se describe en la Figura 2.6; para mantener la simplicidad de la figura sólo se muestran algunos de los atributos de los dos conjuntos de entidades. Como cada entidad *cuenta* participa en una relación con a lo sumo un ejemplar de *cliente*, hacer esta designación de atributos tendría el mismo significado que si se colocase *fecha-acceso* en el conjunto de relaciones *impositor*. Los atributos de un conjunto de relaciones uno a varios se pueden colocar sólo en el conjunto de entidades de la parte «varios» de la relación. Por otra parte, para los conjuntos de entidades uno a uno, los atributos de la relación se pueden asociar con cualquiera de las entidades participantes.

La decisión de diseño de dónde colocar los atributos descriptivos en tales casos —como un atributo de la relación o de la entidad— podría reflejar las características de la empresa que se modela. El diseñador puede elegir mantener *fecha-acceso* como un atributo de *impositor* para expresar explícitamente que ocurre un acceso en el punto de interacción entre los conjuntos de entidades *cliente* y *cuenta*.

La elección de la colocación del atributo es más clara para los conjuntos de relaciones varios a varios. Volviendo al ejemplo, especificamos el caso quizá más realista de *impositor* que es un conjunto de relaciones varios a varios, expresando que un cliente puede tener una o más cuentas, y que una cuenta puede ser mantenida por uno o más clientes. Si se expresa la fecha en que un cliente específico accedió por última vez a una cuenta específica, *fecha-acceso* debe ser un atributo del conjunto de relaciones *impositor*, en lugar de una de las entidades participantes. Si *fecha-acceso* fuese un atributo de *cuenta*, por ejemplo, no se podría determinar

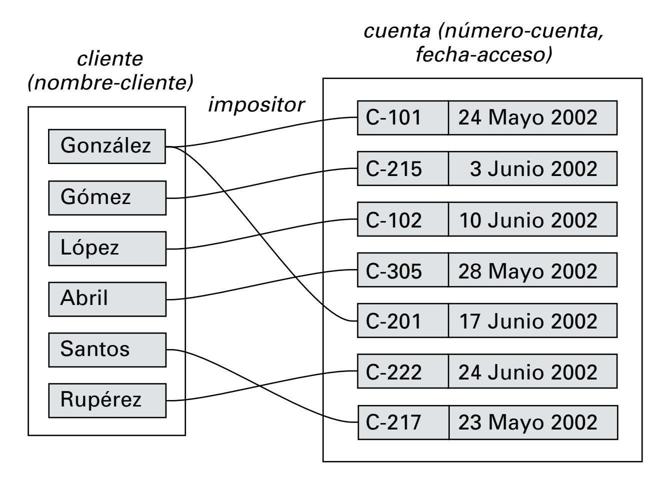


FIGURA 2.6. *Fecha-acceso* como atributo del conjunto de entidades *cuenta*.

qué cliente hizo el acceso más reciente a una cuenta conjunta. Cuando un atributo se determina mediante la combinación de los conjuntos de entidades participantes, en lugar de por cada entidad por separado, ese atributo debe estar asociado con el conjunto de relaciones varios a varios. La colocación de *fecha-acceso* como un atributo de la relación se describe en la Figura 2.7; de nuevo, para mantener la simplicidad de la figura, sólo se muestran algunos de los atributos de los dos conjuntos de entidades.

2.5. DIAGRAMA ENTIDAD-RELACIÓN

Como se vio brevemente en el Apartado 1.4, la estructura lógica general de una base de datos se puede expresar gráficamente mediante un **diagrama E-R**. Los diagramas son simples y claros, cualidades que pueden ser responsables del amplio uso del modelo E-R. Tal diagrama consta de los siguientes componentes principales:

- **Rectángulos**, que representan conjuntos de entidades.
- **Elipses**, que representan atributos.
- **Rombos**, que representan relaciones.
- **Líneas**, que unen atributos a conjuntos de entidades y conjuntos de entidades a conjuntos de relaciones.
- **Elipses dobles**, que representan atributos multivalorados.
- **Elipses discontinuas**, que denotan atributos derivados.
- **Líneas dobles**, que indican participación total de una entidad en un conjunto de relaciones.
- **Rectángulos dobles**, que representan conjuntos de entidades débiles (se describirán posteriormente en el Apartado 2.6).

Considérese el diagrama entidad-relación de la Figura 2.8, que consta de dos conjuntos de entidades, *cliente* y *préstamo*, relacionadas a través de un conjunto de relaciones binarias *prestatario*. Los atributos asociados con *cliente* son *id-cliente*, *nombre-cliente*, *calle-cliente*, y *ciudad-cliente*. Los atributos asociados con *préstamo* son *número-préstamo* e *importe*. Como se muestra en la Figura 2.8, los atributos de un conjunto de entidades que son miembros de la clave primaria están subrayados.

El conjunto de relaciones *prestatario* puede ser varios a varios, uno a varios, varios a uno o uno a uno. Para distinguir entre estos tipos, se dibuja o una línea dirigida (→) o una línea no dirigida (—) entre el conjunto de relaciones y el conjunto de entidades en cuestión.

- Una línea dirigida desde el conjunto de relaciones *prestatario* al conjunto de entidades *préstamo* espe-

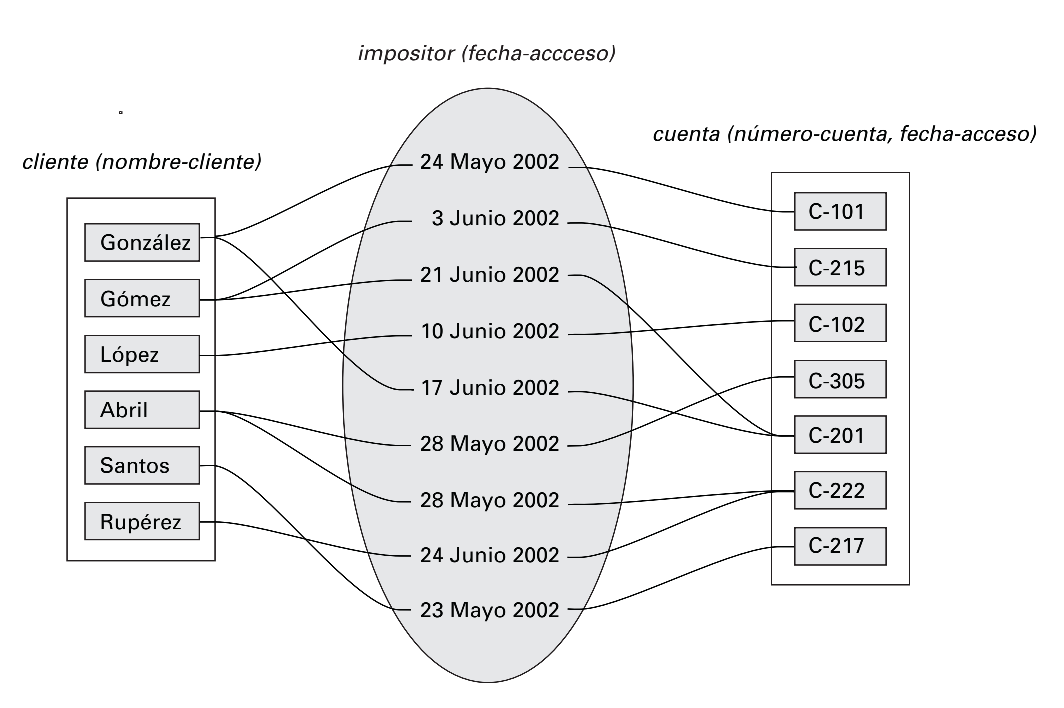


FIGURA 2.7. *Fecha-acceso* como atributo del conjunto de relaciones *impositor*.

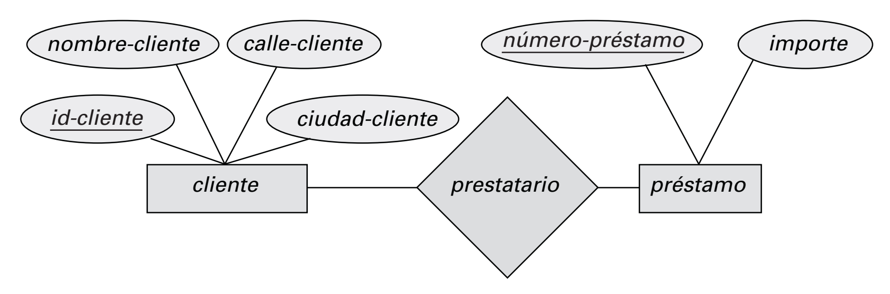


FIGURA 2.8. Diagrama E-R correspondiente a clientes y préstamos.

cifica que *prestatario* es un conjunto de relaciones uno a uno, o bien varios a uno, desde *cliente* a *préstamo*; *prestatario* no puede ser un conjunto de relaciones varios a varios ni uno a varios, desde *cliente* a *préstamo*.

- Una línea no dirigida desde el conjunto de relaciones *prestatario* al conjunto de relaciones *préstamo* especifica que *prestatario* es o bien un conjunto de relaciones varios a varios, o bien uno a varios, desde *cliente* a *préstamo*.

Volviendo al diagrama E-R de la Figura 2.8, se ve que el conjunto de relaciones *prestatario* es varios a varios. Si el conjunto de relaciones *prestatario* fuera uno a varios, desde *cliente* a *préstamo*, entonces la línea desde *prestatario* a *cliente* sería dirigida, con una flecha apuntando al conjunto de entidades *cliente* (Figura 2.9a). Análogamente, si el conjunto de relaciones *pres-*

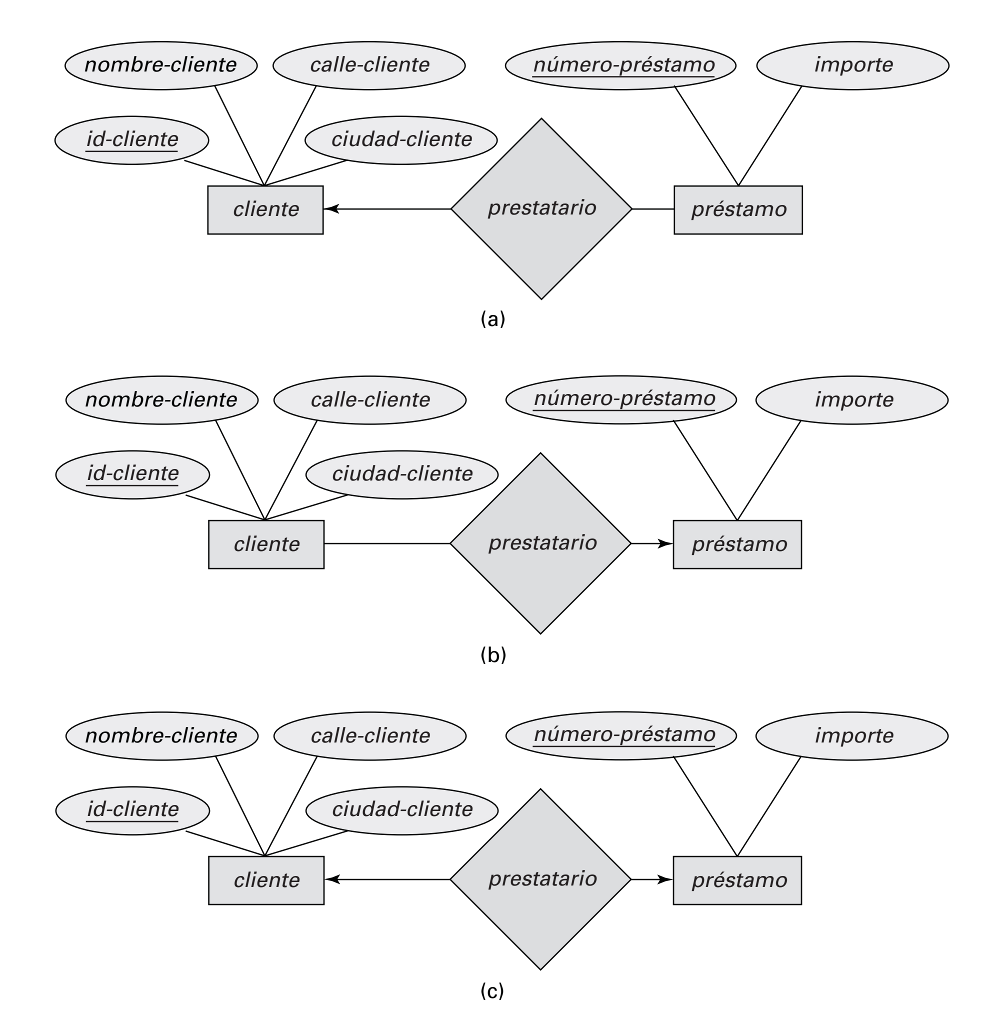


FIGURA 2.9. Relaciones. (a) Uno a varios. (b) Varios a uno. (c) Uno a uno.

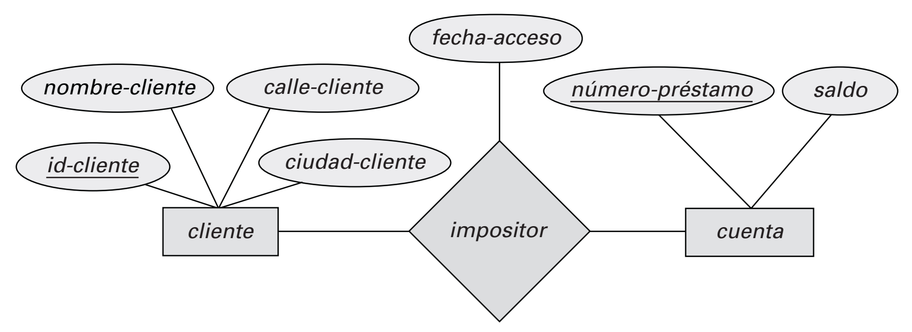


FIGURA 2.10. Diagrama E-R con un atributo unido a un conjunto de relaciones.

tatario fuera varios a uno desde *cliente* a *préstamo*, entonces la línea desde *prestatario* a *préstamo* tendría una flecha apuntando al conjunto de entidades *préstamo* (Figura 2.9b). Finalmente, si el conjunto de relaciones *prestatario* fuera uno a uno, entonces ambas líneas desde *prestatario* tendrían flechas: una apuntando al conjunto de entidades *préstamo* y otra apuntando al conjunto de entidades *cliente* (Figura 2.9c).

Si un conjunto de relaciones tiene también algunos atributos asociados a él, entonces se unen esos atributos a ese conjunto de relaciones. Por ejemplo, en la Figura 2.10, se tiene el atributo descriptivo *fecha-acceso* unido al conjunto de relaciones *impositor* para especificar la fecha más reciente en la que un cliente accedió a esa cuenta.

La Figura 2.11 muestra cómo se pueden representar atributos compuestos en la notación E-R. Aquí, el atributo compuesto *nombre*, con atributos componentes *nombre-pila*, *primer-apellido* y *segundo-apellido* reemplaza al atributo simple *nombre-cliente* de *cliente*. También se muestra el atributo compuesto *dirección*, cuyos atributos componentes son *calle*, *ciudad*, *provincia* y *código-postal*, que reemplaza a los atributos *calle-cliente* y *ciudad-cliente* de *cliente*. El atributo *calle* es por si mismo un atributo compuesto cuyos atributos componentes son *número-calle*, *nombre-calle* y *número-piso*.

La Figura 2.11 también muestra un atributo multivalorado, *número-teléfono*, indicado por una elipse doble, y un atributo derivado *edad*, indicado por una elipse discontinua.

En los diagramas E-R se indican papeles mediante etiquetas en las líneas que unen rombos con rectángulos. En la Figura 2.12 se muestran los indicadores de papeles *director* y *trabajador* entre el conjunto de entidades *empleado* y el conjunto de relaciones *trabaja-para*.

Los conjuntos de relaciones no binarias se pueden especificar fácilmente en un diagrama E-R. La Figura 2.13 consta de tres conjuntos de entidades *cliente*, *trabajo* y *sucursal*, relacionados a través del conjunto de relaciones *trabaja-en*.

Se pueden especificar algunos tipos de relaciones varios a uno en el caso de conjuntos de relaciones no binarias. Supóngase un empleado que tenga a lo sumo un trabajo en cada sucursal (por ejemplo, Santos no puede ser director y auditor en la misma sucursal). Esta restricción se puede especificar con una flecha apuntando a *trabajo* en el borde de *trabaja-en*.

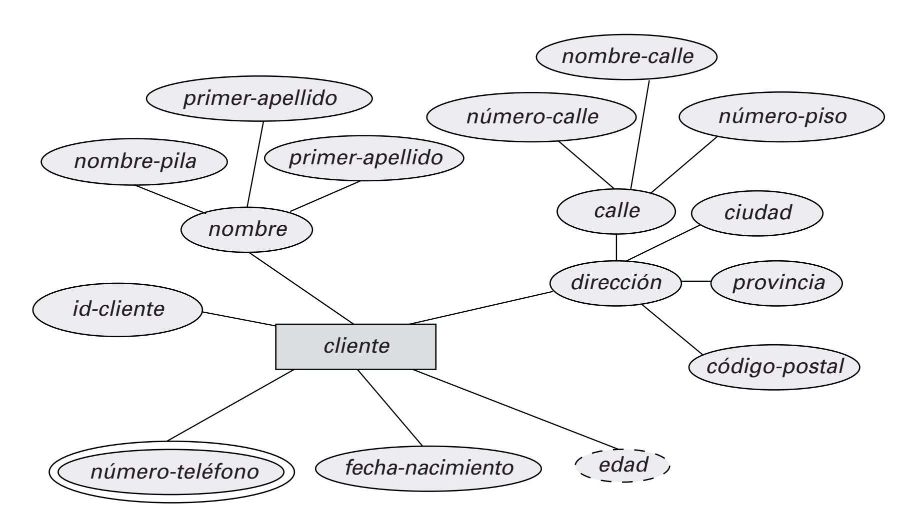


FIGURA 2.11. Diagrama E-R con atributos compuestos, multivalorados y derivados.

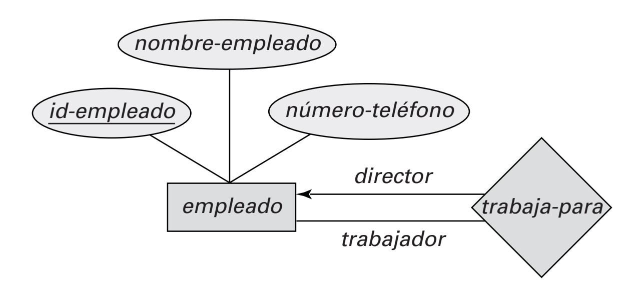


FIGURA 2.12. Diagrama E-R con indicadores de papeles.

Se permite a lo sumo una flecha desde un conjunto de relaciones, ya que un diagrama E-R con dos o más flechas salientes de un conjunto de relaciones no binarias se puede interpretar de dos formas. Supónganse que hay un conjunto de relaciones *R* entre conjuntos de entidades $A_1, A_2, \ldots, A_n$, y las únicas flechas están en los bordes de los conjuntos de entidades $A_{i+1}, A_{i+2}, \ldots, A_n$. Entonces, las dos posibles interpretaciones son:

1. Una combinación particular de entidades de A_1, $A_2, \ldots, A_i$ se puede asociar con a lo sumo una combinación de entidades de $A_{i+1}, A_{i+2}, \ldots, A_n$. Así, la clave primaria de la relación *R* se puede construir por la unión de las claves primarias de $A_1, A_2, \ldots, A_i$.
2. Para cada conjunto de entidades A_k, $i < k \leq = n$, cada combinación de las entidades de los otros conjuntos de entidades se pueden asociar con a lo sumo una entidad de A_k. Cada conjunto $\{A_1, A_2, \ldots, A_{k-1}, A_{k+1}, A_{i+2}, \ldots, A_n\}$, para $i < k \leq = n$, forma entonces una clave candidata.

Cada una de estas interpretaciones se han usado en diferentes libros y sistemas. Para evitar confusión se permite sólo una flecha que salga de un conjunto de relaciones, y así las representaciones son equivalentes. En el Capítulo 7 (Apartado 7.3) se estudia la noción de *dependencias funcionales*, que permiten especificar cualquiera de estas dos interpretaciones sin ambigüedad.

En el diagrama E-R se usan las líneas dobles para indicar que la participación de un conjunto de entidades en un conjunto de relaciones es total; es decir, cada entidad en el conjunto de entidades aparece al menos en una relación en ese conjunto de relaciones. Por ejemplo, considérese la relación *prestamista* entre clientes y préstamos. Una línea doble de *préstamo* a *prestamista*, como en la Figura 2.14, indica que cada préstamo debe tener al menos un cliente asociado.

Los diagramas E-R también proporcionan una forma de indicar restricciones más complejas sobre el número de veces en que cada entidad participa en las relaciones de un conjunto de relaciones. Un segmento entre un conjunto de entidades y un conjunto de relaciones binarias puede tener una cardinalidad mínima y máxima, mostrada de la forma *mín..máx*, donde *mín* es la mínima cardinalidad y *máx* es la máxima. Un valor mínimo de 1 indica una participación total del conjunto de entidades en el conjunto de relaciones. Un valor máximo de 1 indica que la entidad participa de a lo sumo una relación, mientras que un valor máximo de * indica que no hay límite. Nótese que una etiqueta 1..* en un segmento es equivalente a una línea doble.

Por ejemplo, considérese la Figura 2.15. El segmento entre *préstamo* y *prestamista* tiene una restricción de cardinalidad de 1..1, significando que la cardinalidad mínima y máxima son ambas 1. Es decir, cada préstamo debe tener exactamente un cliente asociado. El límite 0..* en el segmento de *cliente* a *prestamista* indica que un cliente puede tener ninguno o varios préstamos. Así, la relación *prestamista* es uno a varios de *cliente* a *préstamo*, y además la participación de *préstamo* en *prestamista* es total.

Es fácil malinterpretar 0..* en el segmento entre *cliente* y *prestamista*, y pensar que la relación *prestamista* es de varios a uno de *cliente* a *préstamo* —esto es exactamente lo contrario a la interpretación correcta.

Si ambos segmentos de una relación binaria tienen un valor máximo de 1, la relación es uno a uno. Si se hubiese especificado un límite de cardinalidad de 1..* en el segmento entre *cliente* y *prestamista*, se estaría diciendo que cada cliente debe tener al menos un préstamo.

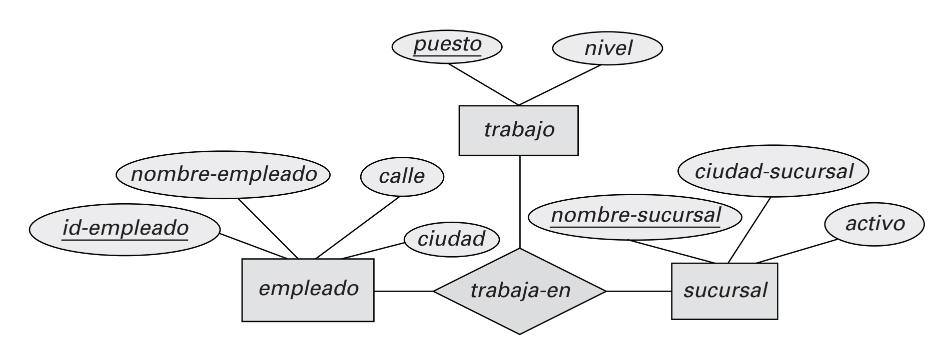


FIGURA 2.13. Diagrama E-R con una relación ternaria.

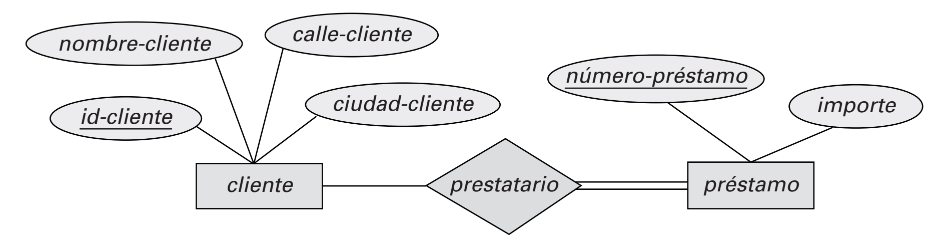


FIGURA 2.14. Participación total de un conjunto de entidades en un conjunto de relaciones.

2.6. CONJUNTOS DE ENTIDADES DÉBILES

Un conjunto de entidades puede no tener suficientes atributos para formar una clave primaria. Tal conjunto de entidades se denomina **conjunto de entidades débiles**. Un conjunto de entidades que tiene una clave primaria se denomina **conjunto de entidades fuertes**.

Como ilustración, considérese el conjunto de entidades *pago*, que tiene los tres atributos: *número-pago, fecha-pago* e *importe-pago*. Los números de pago son generalmente números secuenciales, empezando por 1, generados por separado por cada préstamo. Así, aunque cada entidad *pago* es distinta, los pagos para diferentes préstamos pueden compartir el mismo número de pago. Así, este conjunto de entidades no tiene una clave primaria; es un conjunto de entidades débiles.

Para que un conjunto de entidades débiles tenga sentido, debe estar asociada con otro conjunto de entidades, denominado el **conjunto de entidades identificadoras** o **propietarias**. Cada entidad débil debe estar asociada con una entidad identificadora; es decir, se dice que el conjunto de entidades débiles **depende existencialmente** del conjunto de entidades identificadoras. Se dice que el conjunto de entidades identificadoras es **propietaria** del conjunto de entidades débiles que identifica. La relación que asocia el conjunto de entidades débiles con el conjunto de entidades identificadoras se denomina **relación identificadora**. La relación identificadora es varios a uno del conjunto de entidades débiles al conjunto de entidades identificadoras y la participación del conjunto de entidades débiles en la relación es total.

En nuestro ejemplo, el conjunto de entidades identificador para *pago* es *préstamo*, y la relación *préstamo-pago* que asocia las entidades *pago* con sus correspondientes entidades *préstamo* es la relación identificadora.

Aunque un conjunto de entidades débiles no tiene clave primaria, no obstante se necesita conocer un medio para distinguir todas aquellas entidades del conjunto de entidades que dependen de una entidad fuerte particular. El **discriminante** de un conjunto de entidades débiles es un conjunto de atributos que permite que esta distinción se haga. Por ejemplo, el discriminante del conjunto de entidades débiles *pago* es el atributo *número-pago*, ya que, para cada préstamo, un número de pago identifica de forma única cada pago para ese préstamo. El discriminante de un conjunto de entidades débiles se denomina la *clave parcial* del conjunto de entidades.

La clave primaria de un conjunto de entidades débiles se forma con la clave primaria del conjunto de entidades identificadoras, más el discriminante del conjunto de entidades débiles. En el caso del conjunto de entidades *pago*, su clave primaria es {*número-préstamo*, *número-pago*}, donde *número-préstamo* es la clave primaria del conjunto de entidades identificadoras, es decir, *préstamo*, y *número-pago* distingue las entidades *pago* dentro del mismo préstamo.

El conjunto de entidades identificadoras no debería tener atributos descriptivos, ya que cualquier atributo requerido puede estar asociado con el conjunto de enti-

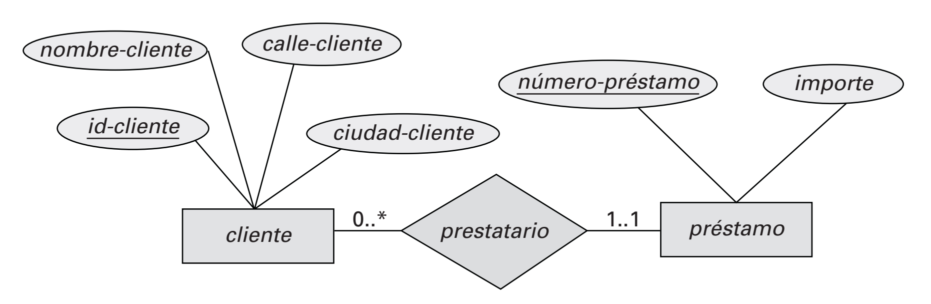


FIGURA 2.15. Límites de cardinalidad en conjuntos de relaciones.

dades débiles (véase la discusión de trasladar los atributos del conjunto de relaciones a los conjuntos de entidades participantes en el Apartado 2.2.1).

Un conjunto de entidades débiles puede participar en relaciones distintas de relaciones identificadoras. Por ejemplo, la entidad *pago* podría participar en una relación con el conjunto de entidades con el conjunto de entidades *cuenta*, identificando la cuenta desde la que se realizó el pago. Un conjunto de entidades débiles puede participar como propietario en una relación identificadora con otro conjunto de entidades débiles. También es posible tener un conjunto de entidades débiles con más de un conjunto de entidades identificadoras. Una entidad débil en concreto podría ser identificada por una combinación de entidades, una de cada conjunto de entidades identificadoras. La clave primaria de la entidad débil consistiría de la unión de las claves primarias de los conjuntos de entidades identificadoras más el discriminante del conjunto de entidades débiles.

Un conjunto de entidades débiles se indica en los diagramas E-R mediante un rectángulo dibujado con una línea doble y la correspondiente relación de identificación mediante un rombo dibujado con línea doble. En la Figura 2.16, el conjunto de entidades débiles *pago* es dependiente del conjunto de entidades fuertes *préstamo* a través del conjunto de relaciones *pago-préstamo*.

La figura ilustra también el uso de líneas dobles para indicar *participación total*; la participación del conjunto de entidades (débiles) *pago* en la relación *pago-préstamo* es total, significando que cada pago debe estar relacionando a través de *pago-préstamo* con alguna cuenta. Finalmente, la flecha desde *pago-préstamo* a *préstamo* indica que cada pago es para un único préstamo. El discriminante del conjunto de entidades débiles también está subrayado, pero con un línea discontinua, en lugar de una continua.

En algunos casos, el diseñador de la base de datos puede elegir expresar un conjunto de entidades débiles como un atributo compuesto multivalorado del conjunto de entidades propietarias. En el ejemplo, esta alternativa requeriría que el conjunto de entidades *préstamo* tuviera un atributo compuesto y multivalorado *pago*, que constara de *número-pago*, *fecha-pago* e *importe-pago*. Un conjunto de entidades débiles se puede modelar más adecuadamente como un atributo si sólo participa en la relación identificadora y si tiene pocos atributos. Alternativamente, una representación de conjunto de entidades débiles será más adecuada para modelar una situación en la que el conjunto participe en otras relaciones además de la relación identificadora y donde el conjunto de entidades débiles tenga muchos atributos.

Como otro de un conjunto de entidades que se puede modelar como un conjunto de entidades débiles considérense las ofertas de asignaturas en una universidad. La misma asignatura se puede ofrecer en diferentes cursos y dentro de un curso puede haber varios grupos para la misma asignatura. Así, se crea un conjunto de entidades débiles *oferta-asignatura*, que depende existencialmente de *asignatura*; las diferentes ofertas de la misma asignatura se identifican por un *curso* y un *número-grupo*, que forma un discriminante pero no una clave primaria.

2.7. CARACTERÍSITCAS DEL MODELO E-R EXTENDIDO

Aunque los conceptos básicos de E-R pueden modelar la mayoría de las características de las bases de datos, algunos aspectos de una base de datos pueden ser más adecuadamente expresados mediante ciertas extensiones del modelo E-R básico. En este apartado se discuten las características E-R extendidas de especialización, generalización, conjuntos de entidades de nivel más alto y más bajo, herencia de atributos y agregación.

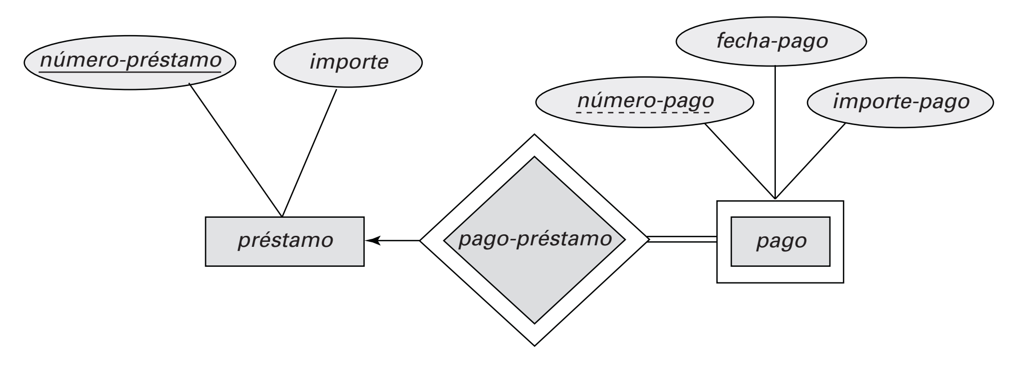


FIGURA 2.16. Diagrama E-R con un conjunto de entidades débiles.

2.7.1. Especialización

Un conjunto de entidades puede incluir subgrupos de entidades que se diferencian de alguna forma de las otras entidades del conjunto. Por ejemplo, un subconjunto de entidades en un conjunto de entidades puede tener atributos que no son compartidos por todas las entidades del conjunto de entidades. El modelo E-R proporciona una forma de representación de estos grupos de entidades distintos.

Considérese el conjunto de entidades *persona* con atributos *nombre*, *calle* y *ciudad*. Una persona puede clasificarse además como:

- *cliente*
- *empleado*

Cada uno de estos tipos de persona se describen mediante un conjunto de atributos que incluyen los atributos del conjunto de entidades *persona* más otros posibles atributos adicionales. Por ejemplo, las entidades *cliente* se pueden describir además mediante el atributo *id-cliente*, mientras que las entidades *empleado* se pueden describir además mediante los atributos *id-empleado* y *sueldo*. El proceso de designación de subgrupos dentro de un conjunto de entidades se denomina **especialización**. La especialización de *persona* permite distinguir entre las personas basándose en si son empleados o clientes.

Como otro ejemplo supóngase que el banco desea dividir las cuentas en dos categorías: cuentas corrientes y cuentas de ahorro. Las cuentas de ahorro necesitan un saldo mínimo, pero el banco establece diferentes tasas de interés a cada cliente, ofreciendo mejores tasas a los clientes favorecidos. Las cuentas corrientes tienen una tasa fija de interés, pero permiten los descubiertos; el importe de descubierto de una cuenta corriente se debe registrar.

El banco podría crear dos especializaciones de *cuenta*, denominadas *cuenta-ahorro* y *cuenta-corriente*. Como se vio anteriormente, las entidades *cuenta* se describen por los atributos *número-cuenta* y *saldo*. El conjunto de entidades *cuenta-ahorro* tendría todos los atributos de *cuenta* y un atributo adicional denominado *tasa-interés*. El conjunto de entidades *cuenta-corriente* tendría todos los atributos de *cuenta* y un atributo adicional *importe-descubierto*.

Se puede aplicar repetidamente la especialización para refinar el esquema de diseño. Por ejemplo, los empleados del banco se pueden clasificar en uno de los siguientes:

- *oficial*
- *cajero*
- *secretaria*

Cada uno de estos tipos de empleado se describe por un conjunto de atributos que incluye todos los atributos del conjunto de entidades *empleado* más otros adicionales. Por ejemplo, las entidades *oficial* se puede describir por el atributo *número-despacho*, las entidades *cajero* por los atributos *número-sección* y *horas-semana*, y las entidades *secretaria* por el atributo *horas-semana*. Además, las entidades *secretaria* pueden participar en una relación *secretaria-de*, que identifica al empleado ayudado por una secretaria.

Un conjunto de entidades se puede especializar por más de una característica distintiva. En el ejemplo, la característica distintiva entre entidades *empleado* es el trabajo que realiza el empleado. Otra especialización coexistente podría estar basada en si la persona es un trabajador temporal o fijo, resultado en los conjuntos de entidades *empleado-temporal* y *empleado-fijo*. Cuando se forma más de una especialización de un conjunto de entidades, una entidad en particular puede pertenecer a varias especializaciones. Por ejemplo, una empleada dada puede ser una empleada temporal y secretaria.

En términos de un diagrama E-R, la especialización se representa mediante un componente *triangular* etiquetado ES, como se muestra en la Figura 2.17. La etiqueta ES representa, por ejemplo, que un cliente «es» una persona. La relación ES se puede llamar también relación **superclase-subclase**. Los conjuntos de entidades de nivel más alto y más bajo se representan como conjuntos de entidades regulares, es decir, como rectángulos que contienen el nombre del conjunto de entidades.

2.7.2. Generalización

El refinamiento a partir de un conjunto de entidades inicial en sucesivos niveles de subgrupos de entidades representa un proceso de diseño **descendente** en el que las distinciones se hacen explícitas. El proceso de diseño puede ser también de una forma **ascendente**, en el que varios conjuntos de entidades se sintetizan en un conjunto de entidades de nivel más alto basado en características comunes. El diseñador de la base de datos puede haber identificado primero el conjunto de entidades *cliente* con los atributos *nombre*, *calle*, *ciudad* e *id-cliente*, y el conjunto de entidades *empleado* con los atributos *nombre*, *calle*, *ciudad*, *id-empleado* y *sueldo*.

Hay similitudes entre el conjunto de entidades *cliente* y el conjunto de entidades *empleado* en el sentido de que tienen varios atributos en común. Esta similitud se puede expresar mediante la **generalización**, que es una relación contenedora que existe entre el conjunto de entidades de *nivel más alto* y uno o más conjuntos de entidades de *nivel más bajo*. En el ejemplo, *persona* es el conjunto de entidades de nivel más alto y los conjuntos de entidades *cliente* y *empleado* son de nivel más bajo. Los conjuntos de entidades de nivel más alto y nivel más bajo también se pueden llamar **superclase** y **subclase**, respectivamente. El conjunto de entidades *persona* es la superclase de las subclases *cliente* y *empleado*.

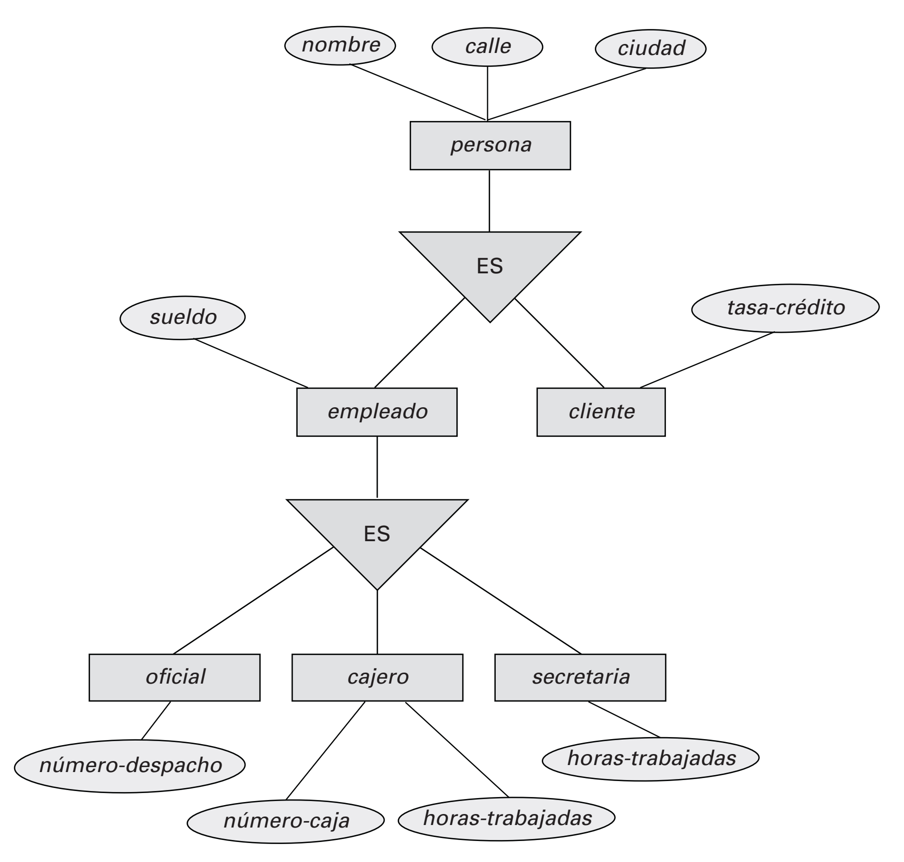


FIGURA 2.17. Especialización y generalización.

Para todos los propósitos prácticos, la generalización es una inversión simple de la especialización. Se aplicarán ambos procesos en combinación en el curso del diseño del esquema E-R para una empresa. En términos del propio diagrama E-R no se distingue entre especialización y generalización. Los niveles nuevos de representación de entidades serán distinguidos (especialización) o sintetizados (generalización) cuando el esquema de diseño llegue a expresar completamente la aplicación de base de datos y los requisitos de uso de la base de datos. Las diferencias entre los dos enfoques se pueden caracterizar mediante su punto de partida y el objetivo global.

La especialización parte de un conjunto de entidades simple; enfatiza las diferencias entre las entidades dentro del conjunto mediante la creación de distintos conjuntos de entidades de nivel más bajo. Estos conjuntos de entidades de nivel más bajo pueden tener atributos, o pueden participar en relaciones que no se aplican a todas las entidades del conjunto de entidades de nivel más alto. Realmente, la razón de que el diseñador aplique la especialización es representar tales características diferentes. Si *cliente* y *empleado* no tuvieran cada una atributos únicos que no tuvieran las entidades *persona* en la que participan, no habría necesidad de especializar el conjunto de entidades *persona*.

La generalización procede de observar que varios conjuntos de entidades que comparten algunas características comunes (se describen mediante los mismos atributos y participan en los mismos conjuntos de relaciones). Basada en sus similitudes, la generalización sintetiza estos conjuntos de entidades en uno solo, el conjunto de entidades de nivel más alto. La generalización se usa para resaltar las similitudes entre los conjuntos de entidades de nivel más bajo y para ocultar las diferencias; también permite economizar la representación para que los atributos compartidos no estén repetidos.

2.7.3. Herencia de atributos

Una propiedad crucial de las entidades de nivel más alto y más bajo creadas mediante especialización y generalización es la **herencia de atributos**. Los atributos de los conjuntos de entidades de nivel más alto se dice que son **heredados** por los conjuntos de entidades de nivel más bajo. Por ejemplo, *cliente* y *empleado* heredan los atributos de *persona*. Así, *cliente* se describe mediante sus atributos *nombre, calle y ciudad* y adicionalmente por el atributo *id-cliente*; *empleado* se describe mediante sus atributos *nombre, calle y ciudad* y adicionalmente por los atributos *id-empleado* y *sueldo*.

Un conjunto de entidades de nivel más bajo (o subclase) también hereda la participación en los conjuntos

de relaciones en los que su entidad de nivel más alto (o superclase) participa. Ambos conjuntos de entidades *oficial*, *cajero* y *secretaria* participan en el conjunto de relaciones *trabaja-para*. La herencia de atributos se aplica en todas las capas de los conjuntos de entidades de nivel más bajo. Los conjuntos de entidades anteriores pueden participar cualquier relación en que participe el conjunto de entidades *persona*.

Si se llega a una porción dada de un modelo E-R mediante especialización o generalización, el resultado es básicamente el mismo:

- Un conjunto de entidades de nivel más alto con atributos y relaciones que se aplican a todos los conjuntos de entidades de nivel más bajo.
- Conjuntos de entidades de nivel más bajo con características distintivas que se aplican sólo en un conjunto de entidades particular.

En lo que sigue, aunque a menudo se hará referencia sólo a la generalización, las propiedades que se discuten pertenecen a ambos procesos.

En la Figura 2.17 se describe una **jerarquía** de conjuntos de entidades. En la figura, *empleado* es un conjunto de entidades de nivel más bajo de *persona* y un conjunto de entidades de nivel más alto de los conjuntos de entidades *oficial*, *cajero* y *secretaria*. En una jerarquía, un conjunto de entidades dado puede estar implicado como un conjunto de entidades de nivel más bajo sólo en una única relación ES. Si un conjunto de entidades es un conjunto de entidades de nivel más bajo en más de una relación ES, entonces el conjunto de entidades tiene **herencia múltiple**, y la estructura resultante se denomina *retículo*.

2.7.4. Restricciones sobre las generalizaciones

Para modelar una empresa más exactamente, el diseñador de la base de datos puede elegir colocar ciertas restricciones en una generalización particular. Un tipo de restricción implica determinar qué entidades pueden ser miembros de un conjunto de entidades de nivel más bajo dado. Tales relaciones de miembros pueden ser algunas de los siguientes:

- **Definido por condición.** En los conjuntos de entidades de nivel más bajo, la relación miembro se evalúa en función de si una entidad satisface o no una condición explícita o predicado. Por ejemplo, asúmase que el conjunto de entidades de nivel más alto *cuenta* tiene el atributo *tipo-cuenta*. Todas las entidades *cuenta* se evalúan según la definición del atributo *tipo-cuenta*. Sólo aquellas entidades que satisfagan la condición *tipo-cuenta* = «cuenta de ahorro» podrán pertenecer al conjunto de entidades de nivel más bajo *cuenta-ahorro*. Todas las entidades que satisfagan la condición *tipo-cuenta* = «cuenta corriente» estarán incluidas en *cuenta-corriente*. Como todas las entidades de nivel más bajo se evalúan en función del mismo atributo (en este caso, *tipo-cuenta*), este tipo de generalización se denomina **definido por atributo**.
- **Definido por el usuario.** Los conjuntos de entidades de nivel más bajo definidos por el usuario no están restringidos mediante una condición de miembro; en cambio, las entidades se asignan a un conjunto de entidades dado por el usuario de la base de datos. Por ejemplo, asúmase que, después de tres meses de empleo, se asignan los empleados del banco a uno de los cuatro grupos de trabajo. Los grupos se representan, por tanto, como cuatro conjuntos de entidades de nivel más bajo del conjunto de entidades de nivel más alto *empleado*. Un empleado dado no se asigna a una entidad grupo automáticamente en términos de una condición que lo defina explícitamente. En su lugar, la asignación al grupo se hace de forma individual por el usuario a cargo de la decisión. Las asignación se implementa mediante una operación que añade una entidad a un conjunto de entidades.

Un segundo tipo de restricciones se define según si las entidades pueden pertenecer a más de un conjunto de entidades de nivel más bajo en una generalización simple. Los conjuntos de entidades de nivel más bajo pueden ser uno de los siguientes:

- **Disjunto.** Una *restricción sobre el carácter disjunto* requiere que una entidad no pertenezca a más de un conjunto de entidades de nivel más bajo. En el ejemplo, una entidad *cuenta* puede satisfacer sólo una condición para el atributo *tipo-cuenta*; una entidad puede ser bien una cuenta de ahorro o bien una cuenta corriente, pero no ambas cosas a la vez.
- **Solapado.** En las *generalizaciones solapadas*, la misma entidad puede pertenecer a más de un conjunto de entidades de nivel más bajo en una generalización simple. Como ilustración, tomando el ejemplo del grupo de trabajo del empleado, asúmase que ciertos directores participen en más de un grupo de trabajo. Un empleado dado puede, por lo tanto, aparecer en más de uno de los conjuntos de entidades grupo que son conjuntos de entidades de nivel más bajo de *empleado*. Así, la generalización es solapada.

  Como otro ejemplo, supóngase la generalización aplicada a los conjuntos de entidades *cliente* y *empleado* conduce a un conjunto de entidades de nivel más alto *persona*. La generalización está solapada si un empleado también puede ser un cliente.

La entidad de nivel más bajo solapada es el caso predeterminado; la restricción sobre el carácter disjunto se debe colocar explícitamente en una generali-

zación (o especialización). Se puede identificar una restricción sobre el carácter disjunto en un diagrama E-R añadiendo la palabra *disjunto* en el símbolo del triángulo.

Una restricción final, la **restricción de completitud** en una generalización o especialización, especifica si un conjunto de entidades de nivel más alto debe pertenecer o no a al menos a uno de los conjuntos de entidades de nivel más bajo en una generalización/especialización. Esta restricción puede ser una de las siguientes:

- **Generalización** o **especialización total.** Cada entidad de nivel más alto debe pertenecer a un conjunto de entidades de nivel más bajo.
- **Generalización** o **especialización parcial.** Algunas entidades de nivel más alto pueden no pertenecer a algún conjunto de entidades de nivel más bajo.

La generalización parcial es la predeterminada. Se puede especificar una generalización total en un diagrama E-R usando una línea doble para conectar el rectángulo que representa el conjunto de entidades de nivel más alto con el símbolo del triángulo (esta notación es similar a la notación de participación total en una relación).

La generalización de *cuenta* es total: todas las entidades *cuenta* deben ser o bien cuentas de ahorro o bien cuentas corrientes. Debido a que el conjunto de entidades de nivel más alto alcanzado a través de la generalización está generalmente compuesta únicamente por aquellas entidades del conjunto de entidades de nivel más bajo, la restricción de completitud para un conjunto de entidades de nivel más alto generalizado es habitualmente total. Cuando la restricción es parcial, la entidad de nivel más alto no aparece necesariamente en el conjunto de entidades de nivel más bajo. Los conjuntos de entidades grupo de trabajo ilustran una especialización parcial. Como los empleados se asignan a grupos sólo después de llevar tres meses en el trabajo, algunas entidades *empleado* pueden no ser miembros de ningún conjunto de entidades grupo de nivel más bajo.

Los conjuntos de entidades equipo se pueden caracterizar más completamente como una especialización de *empleado* parcial y solapada. La generalización de *cuenta-corriente* y *cuenta-ahorro* en *cuenta* es una generalización total y disjunta. Las restricciones de completitud y sobre el carácter disjunto, sin embargo, no dependen una de la otra. Los patrones de restricciones pueden ser también parcial-disjunta y total-solapada.

Se puede ver que ciertos requisitos de inserción y borrado son consecuencia de las restricciones que se aplican a una generalización o especialización dada. Por ejemplo, cuando se coloca una restricción de completitud total, una entidad insertada en un conjunto de entidades de nivel más alto se debe insertar en al menos uno de los conjuntos de entidades de nivel más bajo. Con una restricción de definición por condición, todas las entidades de nivel más alto que satisfacen la condición se deben insertar en el conjunto de entidades de nivel más bajo. Finalmente, una entidad que se borra de un conjunto de entidades de nivel más alto, también se debe borrar de todos los conjuntos de entidades de nivel más bajo asociados a los que pertenezca.

2.7.5. Agregación

Una limitación del modelo E-R es que no resulta posible expresar relaciones entre relaciones. Para ilustrar la necesidad de tales construcciones considérese la relación ternaria *trabaja-en*, que se vio anteriormente, entre *empleado*, *sucursal* y *trabajo* (véase la Figura 2.13). Supóngase ahora que se desean registrar los directores para las tareas realizadas por un empleado en una sucursal; es decir, se desean registrar directores por combinaciones (*empleado*, *sucursal*, *trabajo*). Asúmase que existe una entidad *director*.

Una alternativa para representar esta relación es crear una relación cuaternaria *dirige* entre *empleado*, *sucursal*, *trabajo* y *director* (se necesita una relación cuaternaria; una relación binaria entre *director* y *empleado* no permitiría representar las combinaciones [*sucursal*, *trabajo*] de un empleado que están dirigidas por un director). Al usar los constructores básicos del modelado E-R se obtiene el diagrama E-R de la Figura 2.18 (por simplicidad se han omitido los atributos).

Parece que los conjuntos de relaciones *trabaja-en* y *dirige* se pueden combinar en un único conjunto de relaciones. No obstante, no se deberían combinar, dado que algunas combinaciones *empleado*, *sucursal*, *trabajo* puede que no tengan director.

Hay información redundante en la figura resultante, ya que cada combinación *empleado*, *sucursal*, *trabajo* en *dirige* también lo está en *trabaja-en*. Si el director fuese un valor en lugar de una entidad *director*, se podría hacer que *director* fuese un atributo multivalorado de la relación *trabaja-en*. Pero esto implica que es más difícil (tanto lógicamente como en coste de ejecución) encontrar, por ejemplo, los triples empleado-sucursal-trabajo de los que un director es responsable. Como el director es una entidad *director*, se descarta esta alternativa en cualquier caso.

La mejor forma de modelar una situación como ésta es usar la agregación. La **agregación** es una abstracción a través de la cual las relaciones se tratan como entidades de nivel más alto. Así, para este ejemplo, se considera el conjunto de relaciones *trabaja-en* (que relaciona los conjuntos de entidades *empleado*, *sucursal* y *trabajo*) como un conjunto de entidades de nivel más alto denominado *trabaja-en*. Tal conjunto de entidades se trata de la misma forma que cualquier otro conjunto de entidades. Se puede crear entonces una relación binaria *dirige* entre *trabaja-en* y *director* para representar

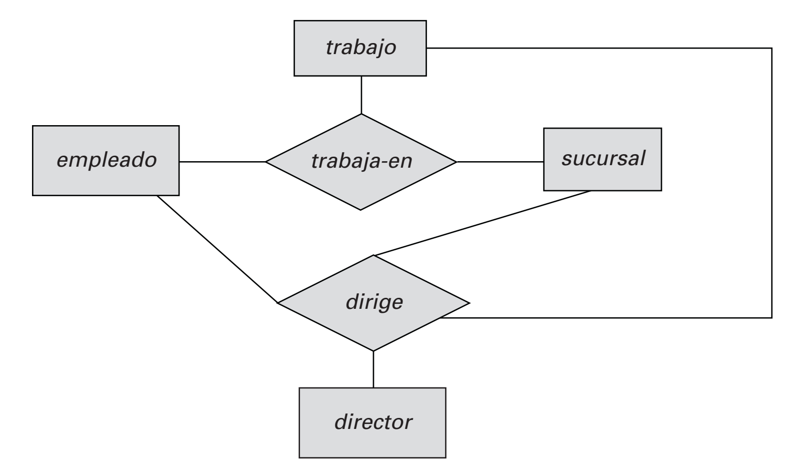


FIGURA 2.18. Diagrama E-R con relaciones redundantes.

quién dirige las tareas. En la Figura 2.19 se muestra una notación para la agregación que se usa habitualmente para esta situación.

2.7.6. Notaciones E-R alternativas

La Figura 2.20 resume el conjunto de símbolos que hemos usado en los diagramas E-R. No hay ningún estándar universal para la notación de los diagramas E-R y diferentes libros y diferente software de diagramas E-R usan notaciones diferentes; la Figura 2.21 indica alguna de las notaciones alternativas que se usan ampliamente. Un conjunto de entidades se puede representar como un cuadro con el nombre fuera, y los atributos listados unos debajo de otros dentro del cuadro. Los atributos clave primaria se indican listándolos en la parte superior, con una línea separándolos de los otros atributos.

Las restricciones de cardinalidad se pueden indicar de varias formas como se muestra en la Figura 2.21. Las etiquetas * y 1 en los arcos que salen de las relaciones se usan a menudo para denotar relaciones varios a varios, uno a uno y varios a uno como se muestra en la figura. En otra notación alternativa de la figura los conjuntos de relaciones se representan por líneas entre conjuntos de entidades sin rombos; sólo se pueden modelar de esta forma las relaciones binarias. Las restricciones de cardinalidad en esta notación se muestran por la notación «pata de gallo», como en la figura.

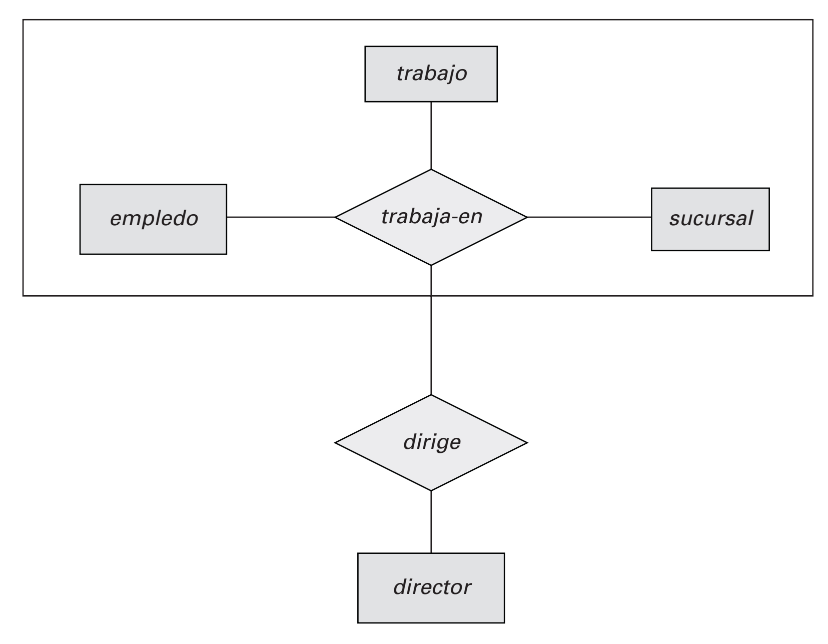


FIGURA 2.19. Diagrama E-R con agregación.

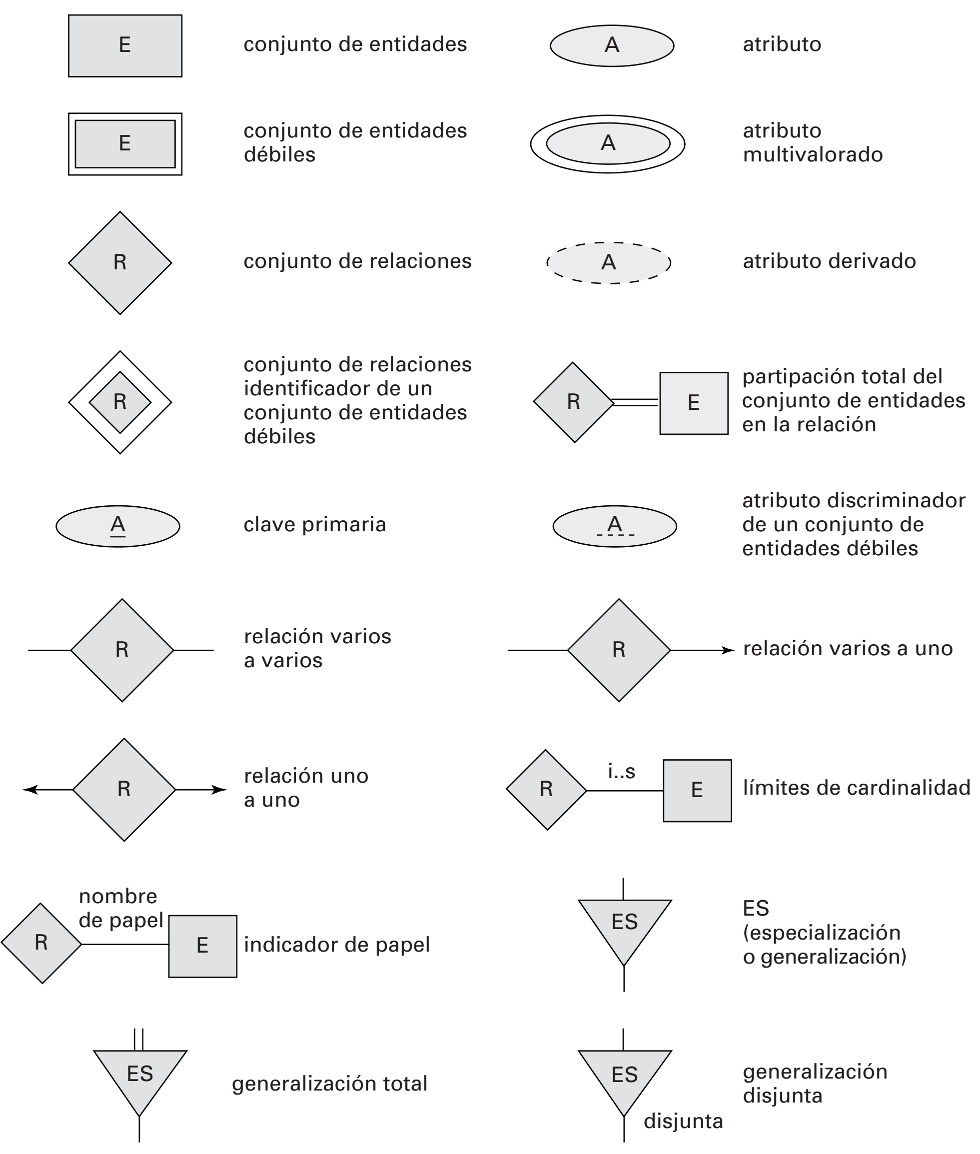


FIGURA 2.20. Símbolos usados en la notación E-R.

2.8. DISEÑO DE UN ESQUEMA DE BASE DE DATOS E-R

El modelo de datos E-R da una flexibilidad sustancial en el diseño de un esquema de bases de datos para modelar una empresa dada. En este apartado se considera cómo un diseñador de bases de datos puede seleccionar entre el amplio rango de alternativas. Entre las decisiones que se toman están las siguientes:

- Si se usa un atributo o un conjunto de entidades para representa un objeto (discutido anteriormente en el Apartado 2.2.1)
- Si un concepto del mundo real se expresa más exactamente mediante un conjunto de entidades o mediante un conjunto de relaciones (Apartado 2.2.2)
- Si se usa una relación ternaria o un par de relaciones binaras (Apartado 2.2.3)
- Si se usa un conjunto de entidades fuertes o débiles (Apartado 2.6); un conjunto de entidades fuertes y sus conjuntos de entidades débiles dependientes se pueden considerar como un «objeto» en la base de datos, debido a que la existencia de las entidades débiles depende de la entidad fuerte
- Si el uso de la generalización (Apartado 2.7.2) es apropiado; la generalización, o una jerarquía de relaciones ES, contribuye a la modularidad por permitir que los atributos comunes de conjuntos de entidades similares se representen en un único lugar en un diagrama E-R
- Si el uso de la agregación (Apartado 2.7.5) es apropiado; la agregación agrupa una parte de un dia-

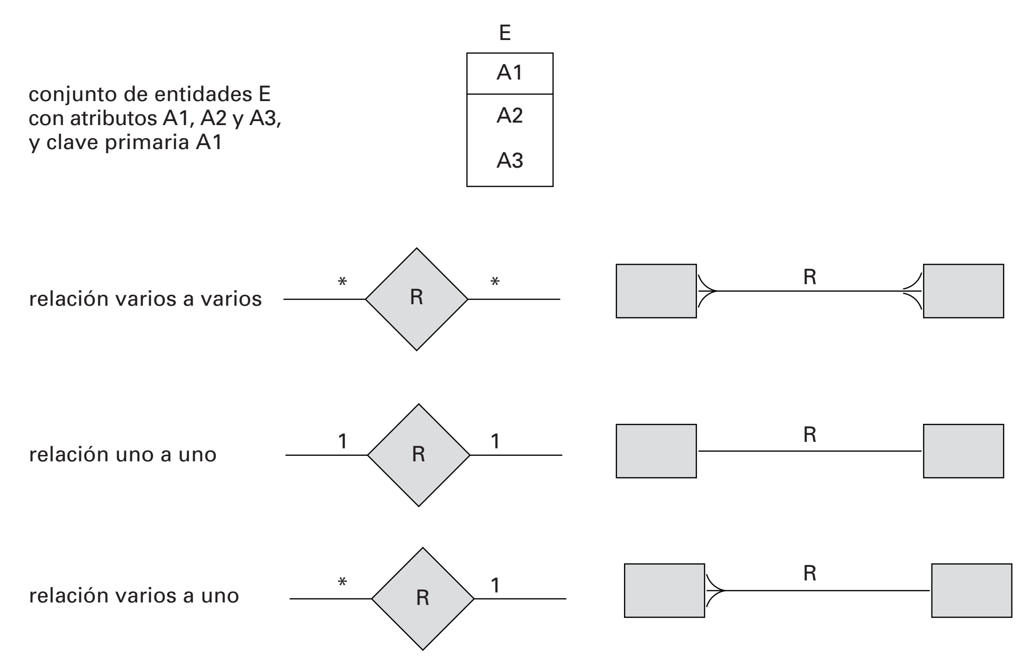


FIGURA 2.21. Notaciones E-R alternativas.

grama E-R en un único conjunto de entidades, permitiendo tratar el conjunto de entidades de la agregación como una unidad única sin importar los detalles de su estructura interna.

Se verá que el diseñador de bases de datos necesita un buen entendimiento de la empresa que se modela para tomar estas decisiones.

2.8.1. Fases de diseño

Un modelo de datos de alto nivel sirve al diseñador de la base de datos para proporcionar un marco conceptual en el que especificar de forma sistemática los requisitos de datos de los usuarios de la base de datos que existen, y cómo se estructurará la base de datos para completar estos requisitos. La fase inicial del diseño de bases de datos, por tanto, es caracterizar completamente las necesidades de datos esperadas por los usuarios de la base de datos. El resultado de esta fase es una *especificación de requisitos del usuario*.

A continuación, el diseñador elige un modelo de datos y, aplicando los conceptos del modelo de datos elegido, traduce estos requisitos a un esquema conceptual de la base de datos. El esquema desarrollado en esta fase de **diseño conceptual** proporciona una visión detallada del desarrollo. Debido a que sólo se ha estudiado el modelo E-R hasta ahora, se usará éste para desarrollar el esquema conceptual. En términos del modelo E-R, el esquema especifica todos los conjuntos de entidades, conjuntos de relaciones, atributos y restricciones de correspondencia. El diseñador revisa el esquema para confirmar que todos los requisitos de datos se satisfacen realmente y no hay conflictos entre sí. También se examina el diseño para eliminar características redundantes. Lo importante en este punto es describir los datos y las relaciones, más que especificar detalles del almacenamiento físico.

Un esquema conceptual completamente desarrollado indicará también los requisitos funcionales de la empresa. En una **especificación de requisitos funcionales** los usuarios describen los tipos de operaciones (o transacciones) que se realizarán sobre los datos. Algunos ejemplos de operaciones son la modificación o actualización de datos, la búsqueda y recuperación de datos específicos y el borrado de datos. En esta fase de diseño conceptual se puede hacer una revisión del esquema para encontrar los requisitos funcionales.

El proceso de trasladar un modelo abstracto de datos a la implementación de la base de datos consta de dos fases de diseño finales. En la **fase de diseño lógico**, el diseñador traduce el esquema conceptual de alto nivel al modelo de datos de la implementación del sistema de base de datos que se usará. El diseñador usa el esquema resultante específico a la base de datos en la siguiente **fase de diseño físico**, en la que se especifican las características físicas de la base de datos. Estas características incluyen la forma de organización de los archivos y las estructuras de almacenamiento interno, que se discutirán en el Capítulo 11.

En este capítulo se tratan sólo los conceptos del modelo E-R usados en la fase de diseño del esquema conceptual. Se ha presentado una breve visión del proceso de diseño de bases de datos para proporcionar un contexto para la discusión del modelo de datos E-R. El diseño de bases de datos recibe un tratamiento completo en el Capítulo 7.

En el Apartado 2.8.2 se aplican las dos fases iniciales de diseño de bases de datos al ejemplo del banco.

Se emplea el modelo de datos E-R para traducir los requisitos de usuario al esquema de diseño conceptual que se describe como un diagrama E-R.

2.8.2. Diseño de base de datos para el banco

Nos centramos ahora en los requisitos de diseño de la base de datos para el banco en más detalle y desarrollamos un diseño más realista, aunque también más complicado, de lo que se ha visto en los ejemplos anteriores. Sin embargo, no se intentará modelar cada aspecto del diseño de la base de datos para un banco; se considerarán sólo unos cuantos aspectos para ilustrar el proceso de diseño de bases de datos.

2.8.2.1. Requisitos de datos

La especificación inicial de los requisitos de usuario se puede basar en entrevistas con los usuarios de la base de datos y en el análisis propio del diseñador del desarrollo. La descripción que surge de esta fase de diseño sirve como base para especificar la estructura conceptual de la base de datos. La siguiente lista describe los principales requisitos del banco:

- El banco está organizado en sucursales. Cada sucursal está ubicada en una ciudad particular y se identifica por un nombre único. El banco supervisa los activos de cada sucursal.
- Los clientes del banco se identifican mediante sus valores de *id-cliente*. El banco almacena cada nombre de cliente, y la calle y ciudad donde viven los clientes. Los clientes pueden tener cuentas y pueden pedir préstamos. Un cliente puede estar asociado con un banquero particular, que puede actuar como responsable de préstamos o banquero personal para un cliente.
- Los empleados del banco se identifican mediante sus valores de *id-empleado*. La administración del banco almacena el nombre y número de teléfono de cada empleado, los nombres de los subordinados del empleado, y el número *id-empleado* del jefe del empleado. El banco también mantiene registro de la fecha de comienzo del contrato del empleado, así como su antigüedad.
- El banco ofrece dos tipos de cuentas: cuentas de ahorro y cuentas corrientes. Las cuentas pueden asociarse a más de un cliente y un cliente puede tener más de una cuenta. Cada cuenta está asignada a un único número de cuenta. El banco mantiene un registro del saldo de cada cuenta y la fecha más reciente en que la cuenta fue accedida por cada cliente que mantiene la cuenta. Además, cada cuenta de ahorro tiene un tipo de interés y para cada cuenta corriente se almacena el descubierto.
- Un préstamo tiene lugar en una sucursal particular y puede estar asociado a uno o más clientes. Un préstamo se identifica mediante un único número de préstamo. Para cada préstamo el banco mantiene registro del importe del préstamo y de los pagos del préstamo. Aunque un número de pago del préstamo no identifica de forma única un pago entre todos los préstamos del banco, un número de pago identifica un pago particular para un préstamo específico. Para cada pago se almacenan la fecha y el importe.

En un desarrollo de un banco real, el banco mantendría información de los abonos y cargos en las cuentas de ahorros y en las cuentas corrientes, igual que se mantiene registro de los pagos para los préstamos. Debido a que los requisitos del modelo para este seguimiento son similares, y para mantener nuestro ejemplo reducido, en este modelo no se mantiene un seguimiento de tales abonos y cargos.

2.8.2.2. Designación de los conjuntos de entidades

La especificación de los requisitos de datos sirve como punto de partida para la construcción de un esquema conceptual para la base de datos. Desde la especificación listada en el Apartado 2.8.2.1 se comienzan a identificar los conjuntos de entidades y sus atributos.

- El conjunto de entidades *sucursal*, con los atributos *nombre-sucursal*, *ciudad-sucursal* y *activo*.
- El conjunto de entidades *cliente*, con los atributos *id-cliente*, *nombre-cliente*, *calle-cliente* y *ciudad-cliente*. Un posible atributo adicional es *nombre-banquero*.
- El conjunto de entidades *empleado*, con los atributos *id-empleado, nombre-empleado, número-teléfono, sueldo y jefe*. Algunas características descriptivas adicionales son el atributo multivalorado *nombre-subordinado*, el atributo base *fecha-comienzo* y el atributo derivado *antigüedad*.
- Dos conjuntos de entidades cuenta —*cuenta-ahorro* y *cuenta-corriente*— con los atributos comunes *número-cuenta* y *saldo*; además, *cuenta-ahorro* tiene el atributo *tipo-interés* y *cuenta-corriente* tiene el atributo *descubierto*.
- El conjunto de entidades *préstamo*, con los atributos *número-préstamo*, *importe* y *sucursal-origen*.
- El conjunto de entidades débiles *pago-préstamo*, con los atributos *número-pago*, *fecha-pago* e *importe-pago*.

2.8.2.3. Designación de los conjuntos de relaciones

Volviendo ahora al esquema de diseño rudimentario del Apartado 2.8.2.2 se especifican los siguientes conjuntos de relaciones y correspondencia de cardinalidades:

- *prestatario*, un conjunto de relaciones varios a varios entre *cliente* y *préstamo*.

- *préstamo-sucursal*, un conjunto de relaciones varios a uno que indica la sucursal en que se ha originado un préstamo. Nótese que este conjunto de relaciones reemplaza al atributo *sucursal-origen* del conjunto de entidades *préstamo*.
- *pago-préstamo*, un conjunto de relaciones uno a varios de *préstamo* a *pago*, que documenta que se ha realizado un pago de un préstamo.
- *impositor*, con el atributo de relación *fecha-acceso*, un conjunto de relaciones varios a varios entre *cliente* y *cuenta*, indicando que un cliente posee una cuenta.
- *banquero-consejero*, con el atributo de relación *tipo*, un conjunto de relaciones varios a uno que expresa que un cliente puede ser aconsejado por un empleado del banco, y que un empleado del banco puede aconsejar a uno o más clientes. Nótese que este conjunto de relaciones ha reemplazado al atributo *nombre-banquero* del conjunto de entidades *cliente*.
- *trabaja-para*, un conjunto de relaciones entre entidades *empleado* con papeles que indican *jefe* y *trabajador*; la correspondencia de cardinalidades expresa que un empleado trabaja para un único jefe, y que un jefe supervisa uno o más empleados. Nótese que este conjunto de relaciones reemplaza el atributo *jefe* de *empleado*.

2.8.2.4. Diagrama E-R

Conforme a lo discutido en el Apartado 2.8.2.3 se presenta ahora el diagrama E-R completo para el ejemplo del banco. En la Figura 2.22 se muestra la representación

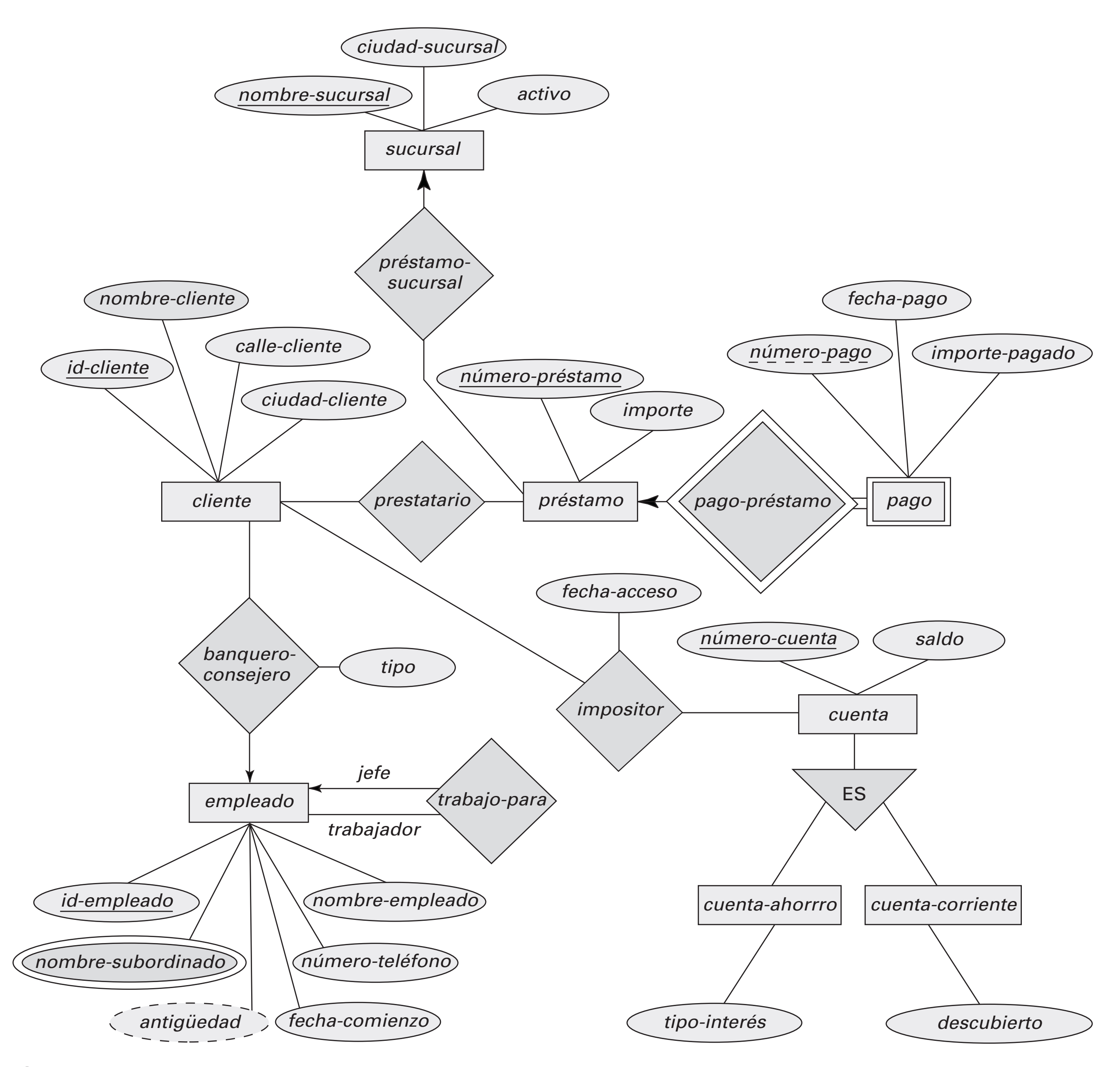


FIGURA 2.22. Diagrama E-R para un banco.

completa de un modelo conceptual de un banco, expresada en términos de los conceptos E-R. El diagrama incluye los conjuntos de entidades, atributos, conjuntos de relaciones, y correspondencia de cardinalidades alcanzados a través del proceso de diseño de los Apartados 2.8.2.1 y 2.8.2.2, y refinados en el Apartado 2.8.2.3.

2.9. REDUCCIÓN DE UN ESQUEMA E-R A TABLAS

Una base de datos que se ajusta a un esquema de bases de datos E-R se puede representar por una colección de tablas. Para cada conjunto de entidades de la base de datos y para cada conjunto de relaciones de la base de datos hay una única tabla a la que se asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondiente. Cada tabla tiene varias columnas, cada una de las cuales tiene un nombre único.

Los modelos E-R y el de bases de datos relacionales son representaciones abstractas y lógicas de empresas del mundo real. Debido a que los dos modelos emplean principios de diseño similares, se puede convertir un diseño E-R en un diseño relacional. Convertir una representación de bases de datos de un diagrama E-R a un formato de tablas es la base para la derivación de un diseño de bases de datos relacional desde un diagrama E-R. Aunque existen diferencias importantes entre una relación y una tabla, una relación se puede considerar informalmente como una tabla de valores.

En este apartado se describe cómo se puede representar un esquema E-R mediante tablas; y en el Capítulo 3 se muestra cómo generar un esquema de bases de datos relacional a partir de un esquema E-R.

Las restricciones especificadas en un diagrama E-R, tales como las claves primarias y las restricciones de cardinalidad, se corresponden con restricciones sobre las tablas generadas a partir del diagrama E-R. Se proporcionan más detalles sobre esta correspondencia en el Capítulo 6 después de describir cómo especificar restricciones sobre tablas.

2.9.1. Representación tabular de los conjuntos de entidades fuertes

Sea *E* un conjunto de entidades fuertes con los atributos descriptivos $a_1, a_2, \ldots, a_n$. Esta entidad se representa mediante una tabla llamada *E* con *n* columnas distintas, cada una de las cuales corresponde a uno de los atributos de *E*. Cada fila de la tabla corresponde a una entidad del conjunto de entidades *E*. (En los apartados 2.9.4 y 2.9.5 se describe cómo manjear los atributos compuestos y multivalorados.)

Como ilustración considérese el conjunto de entidades *préstamo* del diagrama E-R mostrado en la Figura 2.8. Este conjunto de entidades tiene dos atributos: *número-préstamo* e *importe*. Se representa este conjunto de entidades mediante una tabla llamada *préstamo*, con dos columnas, como se muestra en la Figura 2.23. La fila

(P-17,1.000)

número-préstamo	*importe*
P-11	900
P-14	1.500
P-15	1.500
P-16	1.300
P-17	1.000
P-23	2.000
P-93	500

FIGURA 2.23. La tabla *préstamo*.

de la tabla *préstamo* significa que el número de préstamo P-17 tiene un importe de préstamo de 1.000 €. Se puede añadir una nueva entidad a la base de datos insertando una fila en una tabla. También se pueden borrar o modificar las filas.

D_1 denota el conjunto de todos los números de préstamo y D_2 denota el conjunto de todos los saldos. Cualquier fila de la tabla *préstamo* debe consistir en una tupla (v_1, v_2), donde v_1 es un número de préstamo (es decir, v_1 está en el conjunto D_1) y v_2 es un importe (es decir, v_2 está en el conjunto D_2). En general, la tabla *préstamo* contendrá sólo un subconjunto del conjunto de todas las filas posibles. El conjunto de todas las filas posibles de *préstamo* es el *producto cartesiano* de D_1 y D_2, denotado por

$$D_1 \times D_2$$

En general, si se tiene una tabla de *n* columnas, se denota el producto cartesiano de $D_1, D_2, \ldots, D_n$ por

$$D_1 \times D_2 \times \ldots \times D_{n-1} \times D_n$$

Como otro ejemplo considérese el conjunto de entidades *cliente* del diagrama E-R mostrado en la Figura 2.8. Este conjunto de entidades tiene los atributos *id-cliente*, *nombre-cliente*, *calle-cliente* y *ciudad-cliente*. La tabla correspondiente a *cliente* tiene cuatro columnas, como se muestra en la Figura 2.24.

2.9.2. Representación tabular de los conjuntos de entidades débiles

Sea *A* un conjunto de entidades débiles con los atributos $a_1, a_2, \ldots, a_m$. Sea *B* el conjunto de entidades fuertes del que *A* depende. Sea la clave primaria de *B* el conjunto de atributos $b_1, b_2, \ldots, b_n$. Se representa el conjunto de entidades *A* mediante una tabla llamada *A* con una columna por cada uno de los atributos del conjunto:

id-cliente	*nombre-cliente*	*calle-cliente*	*ciudad-cliente*
01.928.374	Gómez	Carretas	Cerceda
18.273.609	Abril	Preciados	Valsaín
19.283.746	González	Arenal	La Granja
24.466.880	Pérez	Carretas	Cerceda
32.112.312	Santos	Mayor	Peguerinos
33.557.799	Fernández	Jazmín	León
33.666.999	Rupérez	Ramblas	León
67.789.901	López	Mayor	Peguerinos
96.396.396	Valdivieso	Goya	Vigo

FIGURA 2.24. La tabla *cliente*.

$$\{a_1, a_2,\ldots,a_m\} \cup \{b_1, b_2,\ldots,b_n\}$$

Como ilustración considérese el conjunto de entidades *pago* mostrado en el diagrama E-R de la Figura 2.16. Este conjunto de entidades tiene tres atributos: *número-pago*, *fecha-pago* e *importe-pago*. La clave primaria del conjunto de entidades *préstamo*, de la que *pago* depende, es *número-préstamo*. Así, *pago* se representa mediante una tabla con cuatro columnas etiquetadas con *número-préstamo*, *número-pago*, *fecha-pago* e *importe-pago*, como se describe en la Figura 2.25.

2.9.3. Representación tabular de los conjuntos de relaciones

Sea R un conjunto de relaciones, sean $a_1, a_2,\ldots,a_m$ el conjunto de atributos formados por la unión de las claves primarias de cada uno de los conjuntos de entidades que participan en R, y sean $b_1, b_2,\ldots,b_n$ los atributos descriptivos de R (si los hay). El conjunto de relaciones se representa mediante una tabla llamada R con una columna por cada uno de los atributos del conjunto:

$$\{a_1, a_2,\ldots,a_m\} \cup \{b_1, b_2,\ldots,b_n\}$$

Como ilustración considérese el conjunto de relaciones *prestatario* del diagrama E-R de la Figura 2.8. Este conjunto de relaciones involucra los dos siguientes conjuntos de entidades:

- *cliente*, con la clave primaria *id-cliente*.
- *préstamo*, con la clave primaria *número-préstamo*.

Debido a que el conjunto de relaciones no tiene atributos, la tabla *prestatario* tiene dos columnas etiquetadas *id-cliente* y *número-préstamo*, como se muestra en la Figura 2.22.

2.9.3.1. Redundancia de tablas

Un conjunto de relaciones uniendo un conjunto de entidades débiles con el correspondiente conjunto de entidades fuertes es un caso especial. Como se hizo notar en el Apartado 2.6, estas relaciones son varios a uno y no tienen atributos descriptivos. Además, la clave primaria de un conjunto de entidades débiles incluye la clave primaria del conjunto de entidades fuertes. En el diagrama E-R de la Figura 2.16, el conjunto de entidades débiles *pago* depende del conjunto de entidades fuertes *préstamo* a través del conjunto de relaciones *pago-préstamo*. La clave primaria de *pago* es {*número-préstamo*, *número-pago*} y la clave primaria de préstamo es {*número-préstamo*}. Como *pago-préstamo* no tiene atributos descriptivos, la tabla para *pago-préstamo* tendría dos columnas, *número-préstamo* y *número-pago*. La tabla para el conjunto de entidades *pago* tiene cuatro columnas, *número-préstamo*, *número-pago*, *fecha-pago* e *importe-pago*. Cada combinación (*número-préstamo*, *número-pago*) en *pago-préstamo* también se encontraría en la tabla *pago*, y viceversa. Por tanto, la tabla *pago-préstamo* es redundante. En general, la tabla para el conjunto de relaciones que une un conjunto de entidades débiles con su correspondiente conjunto de entidades fuertes es redundante y no necesita estar presente en una representación tabular de un diagrama E-R.

número-préstamo	*número-pago*	*fecha-pago*	*importe-pago*
P-11	53	7 junio 2001	125
P-14	69	28 mayo 2001	500
P-15	22	23 mayo 2001	300
P-16	58	18 junio 2001	135
P-17	5	10 mayo 2001	50
P-17	6	7 junio 2001	50
P-17	7	17 junio 2001	100
P-23	11	17 mayo 2001	75
P-93	103	3 junio 2001	900
P-93	104	13 junio 2001	200

FIGURA 2.25. La tabla *pago*.

id-cliente	*número-préstamo*
01.928.374	P-11
01.928.374	P-23
24.466.880	P-93
32.112.312	P-17
33.557.799	P-16
55.555.555	P-14
67.789.901	P-15
96.396.396	P-17

FIGURA 2.26. La tabla *prestatario*.

2.9.3.2. Combinación de tablas

Considérese un conjunto *AB* de relaciones varios a uno del conjunto de entidades *A* al conjunto de entidades *B*. Usando el esquema de construcción de tablas descrito previamente se consiguen tres tablas: *A*, *B* y *AB*. Supóngase además que la participación de *A* en la relación es total; es decir, cada entidad *a* en el conjunto de entidades *A* debe participar en la relación *AB*. Entonces se pueden combinar las tablas *A* y *AB* para formar una única tabla consistente en la unión de las columnas de ambas tablas.

Como ilustración considérese el diagrama E-R de la Figura 2.27. La doble línea del diagrama E-R indica que la participación de *cuenta* en *cuenta-sucursal* es total. Así, una cuenta no puede existir sin estar asociada con una sucursal particular. Además, el conjunto de relaciones *cuenta-sucursal* es varios a uno desde *cuenta* a *sucursal*. Por lo tanto, se puede combinar la tabla para *cuenta-sucursal* con la tabla para *cuenta* y se necesitan sólo las dos tablas siguientes:

- *cuenta*, con los atributos *número-cuenta*, *saldo* y *nombre-cuenta*
- *sucursal*, con los atributos *nombre-sucursal, ciudad-sucursal* y *activo*

En el caso de relaciones uno a uno, la tabla del conjunto de relaciones se puede combinar con las tablas de cualquiera de los conjuntos de entidades. Las tablas se pueden combinar incluso si la participación es parcial usando valores nulos; en el ejemplo anterior se usarían valores nulos para el atributo *nombre-sucursal* para las cuentas que no tengan una sucursal asociada.

2.9.4. Atributos compuestos

Los atributos compuestos se manejan creando un atributo separado para cada uno de los atributos componentes; no se crea una columna separada para el propio atributo compuesto. Supóngase que *dirección* es un atributo compuesto del conjunto de entidades *cliente* y que los componentes de *dirección* son *ciudad* y *calle*. La tabla generada de cliente contendría las columnas *calle-dirección* y *ciudad-dirección*; no hay una columna separada para *dirección*.

2.9.5. Atributos multivalorados

Se ha visto que los atributos en un diagrama E-R generalmente se asocian directamente en columnas para las tablas apropiadas. Los atributos multivalorados, sin embargo, son una excepción; para estos atributos se crean tablas nuevas.

Para un atributo multivalorado *M* se crea una tabla *T* con una columna *C* que corresponde a la clave primaria del conjunto de entidades o conjunto de relaciones del que *M* es atributo. Como ilustración considérese el diagrama E-R de la Figura 2.22. El diagrama incluye el atributo multivalorado *nombre-subordinado*. Para este atributo multivalorado se crea una tabla *nombre-subordinado* con columnas *nombres*, referenciando al atributo *nombre-subordinado* de *empleado*, e *id-empleado*, representado la clave primaria del conjunto de entidades *empleado*. Cada subordinado de un empleado se representa como una única fila en la tabla.

2.9.6. Representación tabular de la generalización

Hay dos métodos diferentes para transformar a forma tabular un diagrama E-R que incluya generalización. Aunque la generalización a la que se va a hacer referencia es la de la Figura 2.17, para simplificar esta discusión se incluye sólo la primera capa de los conjuntos de entidades de nivel más bajo —es decir, *empleado* y *cliente*. Se asume que *nombre* es la clave primaria de *persona*.

1. Crear una tabla para el conjunto de entidades de nivel más alto. Para cada conjunto de entidades

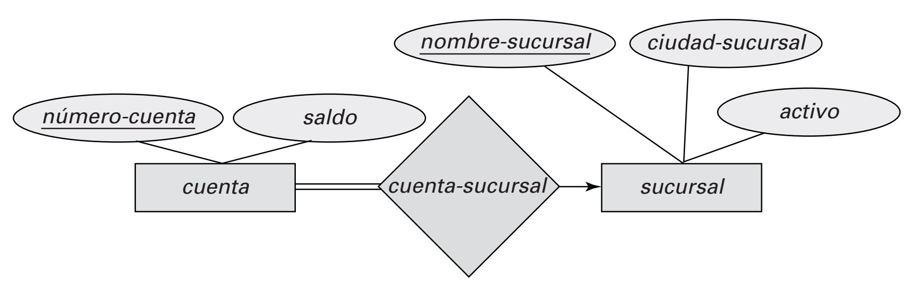


FIGURA 2.27. Diagrama E-R.

de nivel más bajo, crear una tabla que incluya una columna para cada uno de los atributos de ese conjunto de entidades más una columna por cada atributo de la clave primaria del conjunto de entidades de nivel más alto. Así, para el diagrama E-R de la Figura 2.15, se tienen tres tablas:

- *persona*, con atributos *nombre, calle* y *ciudad*
- *empleado*, con atributos *nombre* y *salario*
- *cliente*, con atributos *nombre, límite-crédito*

2. Es posible una representación alternativa si la generalización es disjunta y completa —es decir, si no hay ninguna entidad que sea miembro de dos conjuntos de entidades de menor nivel directamente debajo de un conjunto de entidades de nivel más alto, y si cada entidad del conjunto de entidades de nivel más alto también pertenece a uno de los conjuntos de entidades de nivel más bajo. Aquí no se crea una tabla para el conjunto de entidades de nivel más alto. En su lugar, para cada conjunto de entidades de nivel más bajo se crea una tabla que incluya una columna por cada atributo del conjunto de entidades más una columna por *cada* atributo del conjunto de entidades de nivel más alto. Entonces, para el diagrama E-R de la Figura 2.15 se tienen dos tablas.

   - *empleado*, con atributos *nombre, calle, ciudad* y *sueldo*
   - *cliente*, con atributos *nombre, calle, ciudad* y *límite-crédito*

   Las relaciones *cuenta-ahorro* y *cuenta-corriente* correspondientes a esas tablas tienen *número-cuenta* como clave primaria.

Si se usara el segundo método para una generalización solapada, algunos valores se almacenarían varias veces innecesariamente. Por ejemplo, si una persona es tanto empleado como cliente, los valores de *calle* y *ciudad* se almacenarían dos veces. Si la generalización no fuera completa (es decir, si alguna persona no fuera ni empleado no cliente) entonces se necesitaría una tabla extra *persona* para representarlos.

2.9.7. Representación tabular de la agregación

Transformar a forma tabular un diagrama E-R que incluya agregación es sencillo. Considérese el diagrama de la Figura 2.19. La tabla para el conjunto de relaciones *dirige* entre la agregación de *trabaja-en* y el conjunto de entidades *director* incluye una columna para cada atributo de la clave primaria del conjunto de entidades *director* y del conjunto de relaciones *trabaja-en*. También incluiría una columna para los atributos descriptivos, si los hubiera, del conjunto de relaciones *dirige*. Por tanto, se transforman los conjuntos de relaciones y los conjuntos de entidades dentro de la entidad agregada.

2.10. EL LENGUAJE DE MODELADO UNIFICADO UML (*Unified Modeling Language*)**

Los diagramas entidad-relación ayudan a modelar el componente de representación de datos de un sistema software. La representación de datos, sin embargo, sólo forma parte de un diseño completo de un sistema. Otros componentes son modelos de interacción del usuario con el sistema, especificación de módulos funcionales del sistema y su interacción, etc. El **lenguaje de modelado unificado** (UML, *Unified Modeling Language*) es un estándar propuesto para la creación de especificaciones de varios componentes de un sistema software. Algunas de las partes de UML son:

- **Diagrama de clase.** Un diagrama de clase es similar a un diagrama E-R. Más adelante en este apartado se mostrarán algunas características de los diagramas de clase y cómo se corresponden con los diagramas E-R.
- **Diagrama de caso de uso.** Los diagramas de caso de uso muestran la interacción entre los usuarios y el sistema, en particular los pasos de las tareas que realiza el usuario (tales como prestar dinero o matricularse de una asignatura).
- **Diagrama de actividad.** Los diagramas de actividad describen el flujo de tareas entre varios componentes de un sistema.
- **Diagrama de implementación.** Los diagramas de implementación muestran los componentes del sistema y sus interconexiones tanto en el nivel del componente software como el hardware.

Aquí no se intentará proporcionar un tratamiento detallado de las diferentes partes de UML. Véanse las notas bibliográficas para encontrar referencias de UML. En su lugar se ilustrarán algunas características de UML mediante ejemplos.

La Figura 2.28 muestra varios constructores de diagramas E-R y sus constructores equivalentes de los diagramas de clase UML. Más abajo se describen estos constructores. UML muestra los conjuntos de entidades como cuadros y, a diferencia de E-R, muestra los atributos dentro del cuadro en lugar de como elipses sepa-

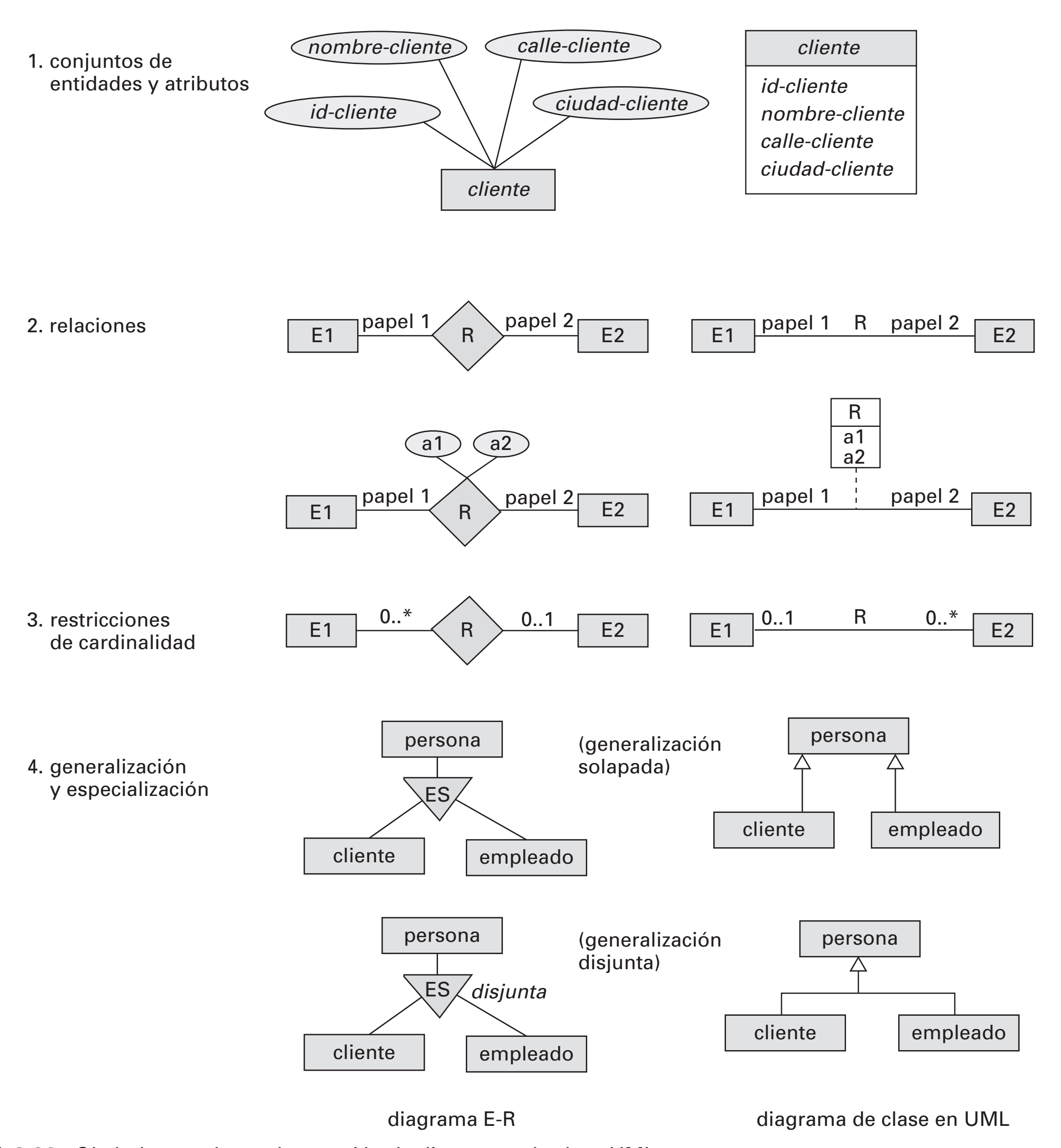


FIGURA 2.28. Símbolos usados en la notación de diagramas de clase UML.

radas. UML modela realmente objetos, mientras que E-R modela entidades. Los objetos son como entidades y tienen atributos, pero además proporcionan un conjunto de funciones (denominadas métodos) que se pueden invocar para calcular valores en términos de los atributos de los objetos, o para modificar el propio objeto. Los diagramas de clase pueden describir métodos además de atributos. Los objetos se tratan en el Capítulo 8.

Los conjuntos de relaciones binarias se representan en UML dibujando simplemente una línea que conecte los conjuntos de entidades. Se escribe el nombre del conjunto de relaciones adyacente a la línea. También se puede especificar el papel que juega un conjunto de entidades en un conjunto de relaciones escribiendo el nombre del papel en un cuadro, junto con los atributos del conjunto de relaciones, y conectar el cuadro con una línea discontinua a la línea que describe el conjunto de relaciones. Este cuadro se puede tratar entonces como un conjunto de entidades, de la misma forma que una agregación en los diagramas E-R puede participar en relaciones con otros conjuntos de entidades.

La relaciones no binarias no se pueden representar directamente en UML —se deben convertir en relaciones binarias por la técnica que se describió en el Apartado 2.4.3.

Las restricciones de cardinalidad se especifican en UML de la misma forma que en los diagramas E-R, de la forma *i*..*s*, donde *i* denota el mínimo y *h* el máximo número de relaciones en que puede participar una entidad. Sin embargo, se debería ser consciente que la ubicación de las restricciones es exactamente el inverso de la ubicación de las restricciones en los diagramas E-R, como muestra la Figura 2.28. La restricción 0..* en el lado *E*2 y 0..1 en el lado *E*1 significa que cada entidad

*E*2 puede participar a lo sumo en una relación, mientras que cada entidad *E*1 puede participar en varias relaciones; en otras palabras, la relación es varios a uno de *E*2 a *E*1.

Los valores como 1 o * se pueden escribir en los arcos; el valor 1 sobre un arco se trata equivalentemente como 1..1, mientras que * es equivalente a 0..*.

La generalización y especialización se representan en el diagrama E-R conectando conjuntos de entidades por una línea con un triángulo al final correspondiente al conjunto de entidades más general. Por ejemplo, el conjunto de entidades *persona* es una generalización de *cliente* y *empleado*. Los diagramas UML también pueden representar explícitamente las restricciones de generalizaciones disjuntas y solapadas. La Figura 2.28 muestra generalizaciones disjuntas y solapadas de *cliente* y *empleado* a *persona*. Recuérdese que se la generalización de *cliente* / *empleado* a *persona* es disjunta, y significa que ninguna entidad puede ser a la vez un *cliente* y un *empleado*. Una generalización solapada permite que una persona sea tanto *cliente* como *empleado*.

2.11. RESUMEN

- El modelo de datos **entidad-relación (E-R)** se basa en una percepción del mundo real consistente en un conjunto de objetos básicos llamados **entidades** y en **relaciones** entre esos objetos.
- El modelo está pensado principalmente para el proceso de diseño de la base de datos. Fue desarrollado para facilitar el diseño permitiendo la especificación de un **esquema de la empresa**. Tal esquema representa la estructura lógica general de la base de datos. Esta estructura general se puede expresar gráficamente mediante un **diagrama E-R**.
- Una **entidad** es un objeto que existe y es distinguible de otros objetos. Se expresa la distinción asociando con cada entidad un conjunto de atributos que describen el objeto.
- Una **relación** es una asociación entre diferentes entidades. Un **conjunto de relaciones** es una colección de relaciones del mismo tipo y un **conjunto de entidades** es una colección de entidades del mismo tipo.
- La **correspondencia de cardinalidades** expresa el número de entidades a las que otra entidad se puede asociar a través de un conjunto de relaciones.
- Una **superclave** de un conjunto de entidades es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar unívocamente una entidad en un conjunto de entidades. Se elige una superclave mínima para cada conjunto de entidades de entre sus superclaves; la superclave mínima se denomina la **clave primaria** del conjunto de entidades. Análogamente, un conjunto de relaciones es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar unívocamente una relación en un conjunto de relaciones. De igual forma se elige una superclave mínima para cada conjunto de relaciones de entre todas sus superclaves; ésta es la clave primaria del conjunto de relaciones.
- Un conjunto de entidades que no tiene suficientes atributos para formar una clave primaria se denomina **conjunto de entidades débiles**. Un conjunto de entidades que tiene una clave primaria se denomina **conjunto de entidades fuertes**.
- La **especialización** y la **generalización** definen una relación de contenido entre un conjunto de entidades de nivel más alto y uno o más conjuntos de entidades de nivel más bajo. La especialización es el resultado de tomar un subconjunto de un conjunto de entidades de nivel más alto para formar un conjunto de entidades de nivel más bajo. La generalización es el resultado de tomar la unión de dos o más conjuntos disjuntos de entidades (de nivel más bajo) para producir un conjunto de entidades de nivel más alto. Los atributos de los conjuntos de entidades de nivel más alto los heredan los conjuntos de entidades de nivel más bajo.
- La **agregación** es una abstracción en la que los conjuntos de relaciones (junto con sus conjuntos de entidades asociados) se tratan como conjuntos de entidades de nivel más alto, y pueden participar en las relaciones.
- Las diferentes características del modelo E-R ofrecen al diseñador de bases de datos numerosas decisiones de cómo representar mejor la empresa que se modela. Los conceptos y objetos pueden, en ciertos casos, representarse mediante entidades, relaciones o atributos. Ciertos aspectos de la estructura global de la empresa se pueden describir mejor usando conjuntos de entidades débiles, generalización, especialización o agregación. A menudo el diseñador debe sopesar las ventajas de un modelo simple y compacto frente a otros más precisos pero más completos.
- Una base de datos que se representa en un diagrama E-R se puede representar mediante una colección de tablas. Para cada conjunto de entidades y para cada conjunto de relaciones de la base de datos hay una única tabla a la que se le asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondiente. Cada tabla tiene un número de columnas, cada una de las cuales tiene un nombre único. La conversión de una representación de base de datos en un diagrama E-R a un formato de tabla se basa en la deri-

vación de un diseño de bases de datos relacional desde un diagrama E-R.

- El **lenguaje de modelado unificado (UML)** proporciona un medio gráfico de modelar varios componentes de un sistema software. El componente diagrama de clase de UML se basa en diagramas E-R. Sin embargo, hay algunas diferencias entre ambos que se deben tener presentes.

TÉRMINOS DE REPASO

- Agregación
- Atributo derivado
- Atributos
- Atributos descriptivos
- Atributos monovalorados y multivalorados
- Atributos simples y compuestos
- Conjunto de entidades
- Conjunto de relaciones
- Conjunto de relaciones binario
- Conjunto de relaciones recursivo
- Conjuntos de entidades débiles y fuertes
  — Atributos discriminantes
  — Relaciones identificadoras
- Correspondencia de cardinalidad:
  — Relación uno a uno
  — Relación uno a varios
  — Relación varios a uno
  — Relación varios a varios
- Diagrama E-R
- Dominio
- Entidad
- Especialización y generalización
  — Superclase y subclase
  — Herencia de atributos
  — Herencia simple y múltiple
  — Pertenencia definida por condición y definida por el usuario
  — Generalización disjunta y solapada
- Grado de un conjunto de relaciones
- Lenguaje de modelado unificado (UML)
- Modelo de datos entidad-relación
- Papel
- Participación
  — Participación total
  — Participación parcial
- Relación
- Restricción de completitud
  — Generalización total y parcial
- Superclave, clave candidata y clave primaria
- Valor nulo

EJERCICIOS

2.1. Explíquense las diferencias entre los términos clave primaria, clave candidata y superclave.

2.2. Constrúyase un diagrama E-R para una compañía de seguros de coches cuyos clientes poseen uno o más coches. Cada coche tiene asociado un número de cero a cualquier valor que almacena el número de accidentes.

2.3. Constrúyase un diagrama E-R para un hospital con un conjunto de pacientes y un conjunto de médicos. Asóciese con cada paciente un registro de las diferentes pruebas y exámenes realizados.

2.4. Una oficina de registro de una universidad mantiene datos acerca de las siguientes entidades: (a) asignaturas, incluyendo el número, título, programa, y prerrequisitos; (b) ofertas de asignaturas, incluyendo número de asignatura, año, semestre, número de sección, profesor(es), horarios y aulas; (c) estudiantes, incluyendo id-estudiante, nombre y programa; y (d) profesores, incluyendo número de identificación, nombre, departamento y título. Además, la matrícula de los estudiantes en asignaturas y las notas concedidas a estudiantes en cada asignatura en la que están matriculados se deben modelar adecuadamente.

Constrúyase un diagrama E-R para la oficina de registro. Documéntense todas las decisiones que se hagan acerca de restricciones de correspondencia.

2.5. Considérese una base de datos usada para registrar las notas que obtienen los estudiantes en diferentes exámenes de diferentes ofertas de asignaturas.

a. Constrúyase un diagrama E-R que modele exámenes como entidades y use una relación ternaria para esta base de datos.

b. Constrúyase un diagrama E-R alternativo que use sólo una relación binaria entre *estudiantes* y *ofertas-asignaturas*. Asegúrese de que sólo existe una relación entre un par determinado estudiante y oferta-asignatura y de que aún se pueden representar las notas que obtiene un estudiante en diferentes exámenes de una oferta de una asignatura.

2.6. Constrúyanse tablas apropiadas para cada uno de los diagramas E-R de los Ejercicios 2.2 al 2.4.

2.7. Diséñese un diagrama E-R para almacenar los logros de su equipo deportivo favorito. Se deberían almacenar los partidos jugados, los resultados de cada partido, los jugadores de cada partido y las estadísticas individuales de cada jugador para cada partido. Las estadísticas de resumen se deberían modelar como atributos derivados.

2.8. Extiéndase el diagrama E-R del ejercicio anterior para almacenar la misma información para todos los equipos de una liga.

2.9. Explíquense las diferencias entre conjunto de entidades débiles y fuertes.

2.10. Se puede convertir cualquier conjunto de entidades débiles en un conjunto de entidades fuertes simplemente añadiendo los atributos apropiados. ¿Por qué, entonces, se tienen conjuntos de entidades débiles?

2.11. Defínase el concepto de agregación. Propónganse ejemplos para los que este concepto es útil.

2.12. Considérese el diagrama de la Figura 2.29, que modela una librería en línea.

a. Lístense los conjuntos de entidades y sus claves primarias.

b. Supóngase que la librería añade casetes de música y discos compactos a su colección. El mismo elemento musical puede estar presente en formato de casete o de disco compacto con diferentes precios. Extiéndase el diagrama E-R para modelar esta adición, ignorando el efecto sobre las cestas de la compra.

c. Extiéndase ahora el diagrama E-R usando generalización para modelar el caso en que una cesta de la compra pueda contener cualquier combinación de libros, casetes de música o discos compactos.

2.13. Considérese un diagrama E-R en el que el mismo conjunto de entidades aparece varias veces. ¿Por qué está permitida esta redundancia, una mala práctica que se debería evitar siempre que sea posible?

2.14. Considérese una base de datos de una universidad para la planificación de las aulas para los exámenes finales. Esta base de datos se modelaría mediante un único conjunto de entidades *examen*, con atributos *nombre-asignatura*, *número-sección*, *número-aula* y *hora*. Alternativamente se podrían definir uno o más conjuntos de entidades, con conjuntos de relaciones para sustituir algunos de los atributos del conjunto de entidades *examen*, como

- *asignatura* con atributos *nombre, departamento* y *número-a*
- *sección* con atributos *número-s* y *matriculados*, que es un conjunto de entidades débiles dependiente de *curso*.
- *aula* con atributos *número-a, capacidad* y *edificio*.

a. Muéstrese en un diagrama E-R el uso de los tres conjuntos de entidades adicionales listados.

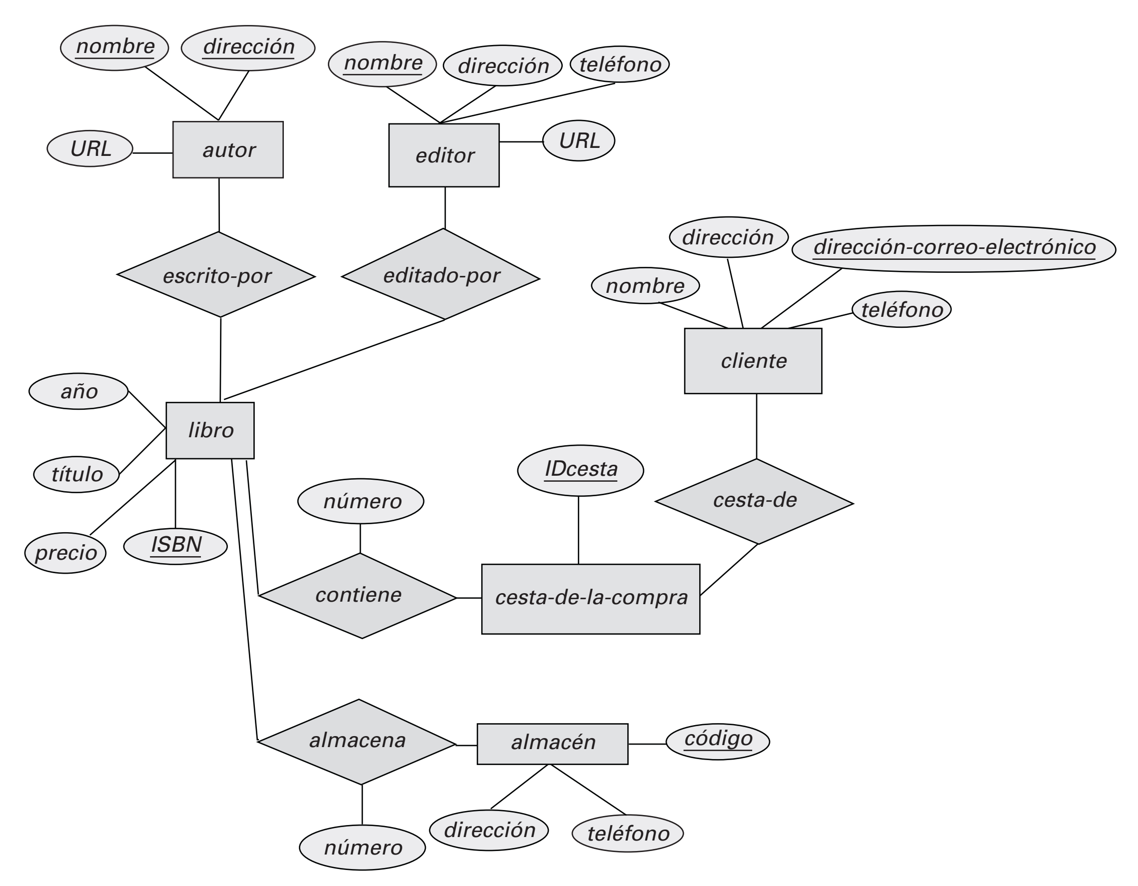


FIGURA 2.29. Diagrama E-R para el Ejercicio 2.12.

b. Explíquense las características que influirían en la decisión de incluir o no incluir cada uno de los conjuntos de entidades adicionales.

2.15. Cuando se diseña un diagrama E-R para un desarrollo particular se tienen varias alternativas entre las que hay que decidir.

a. ¿Qué criterio se deberá considerar para hacer la elección apropiada?

b. Diséñense tres alternativas de diagrama E-R para representar la oficina de registro de la universidad del Ejercicio 2.4. Lístense las ventajas de cada uno. Decídase por una de las alternativas.

2.16. Un diagrama E-R se puede ver como un grafo. ¿Qué significan los siguientes términos de estructura en un esquema de desarrollo?

a. El grafo es inconexo.

b. El grafo es acíclico.

2.17. En el Apartado 2.4.3 se representó una relación ternaria (Figura 2.30a) usando relaciones binarias, como se muestra en la Figura 2.30b. Considérese la alternativa mostrada en la Figura 2.30c. Discútanse las ventajas relativas a estas dos representaciones alternativas entre una relación ternaria y relaciones binarias.

2.18. Considérese la representación de una relación ternaria usando relaciones binarias como se describió en el Apartado 2.4.3 (mostrado en la figura 2.30b).

a. Muéstrese un ejemplar simple de *E*, *A*, *B*, *C*, R_A, R_B y R_C que no puedan corresponder a ningún ejemplar de *A*, *B*, *C* y *R*.

b. Modifíquese el diagrama E-R de la Figura 2.30b para introducir restricciones que garanticen que cualquier ejemplar *E*, *A*, *B*, *C*, R_A, R_B y R_C que satisfaga las restricciones corresponda a un ejemplar de *A*, *B*, *C* y *R*.

c. Modifíquese la traducción de arriba para manejar restricciones de participación total sobre las relaciones ternarias.

d. La representación de arriba requiere que se cree un atributo clave primaria para *E*. Muéstrese cómo tratar *E* como un conjunto de entidades débiles de forma que no se requiera un atributo clave primaria.

2.19. Un conjunto de entidades débiles siempre se puede convertir en un conjunto de entidades fuertes añadiéndole a sus atributos los atributos clave primaria de su conjunto de entidades identificadoras. Descríbase qué tipo de redundancia resultaría si se hiciese así.

2.20. Diséñese una jerarquía de especialización-generalización para las ventas de una compañía de vehículos a motor. La compañía vende motocicletas, coches de pasajeros, furgonetas y autobuses. Justifíquese la colocación de los atributos en cada nivel de la jerarquía. Explíquese por qué se deberían colocar en un nivel más alto o más bajo.

2.21. Explíquese la distinción entre las restricciones de diseño definidas por condición y las definidas por el usuario. ¿Cuáles de estas restricciones se pueden comprobar automáticamente? Explíquese la respuesta.

2.22. Explíquese la distinción entre las restricciones disjuntas y solapadas.

2.23. Explíquese la distinción entre las restricciones totales y parciales.

2.24. En la Figura 2.31 se muestra una estructura reticular de generalización y especialización. Para los conjuntos de entidades *A*, *B* y *C* explíquese cómo se heredan los atributos desde los conjuntos de entidades de nivel más alto *X* e *Y*. Discútase cómo manejar el caso en que un atributo de *X* tiene el mismo nombre que un atributo de *Y*.

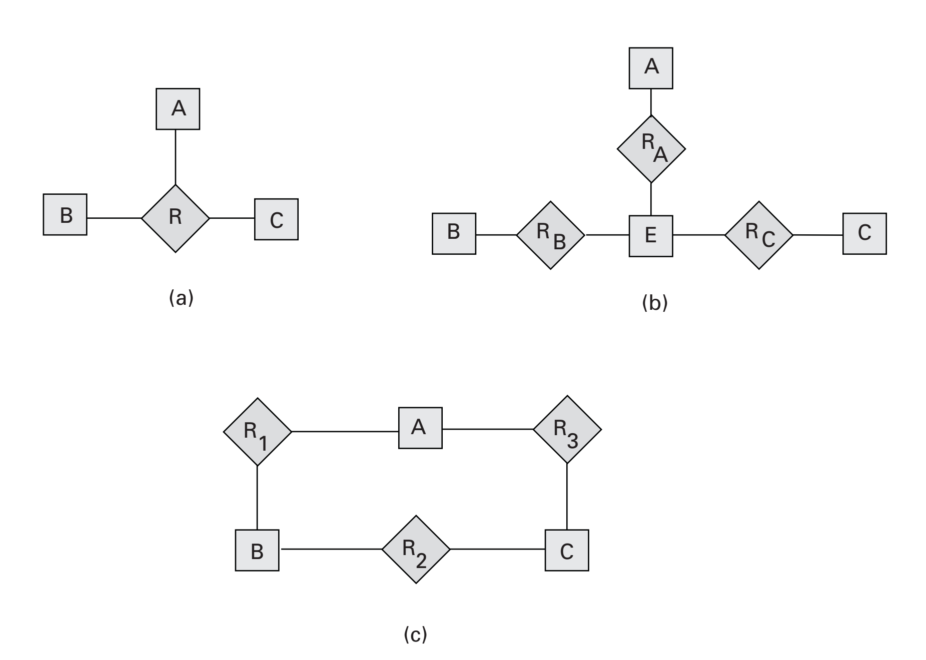


FIGURA 2.30. Diagrama E-R para el Ejercicio 2.17 (no se muestran los atributos).

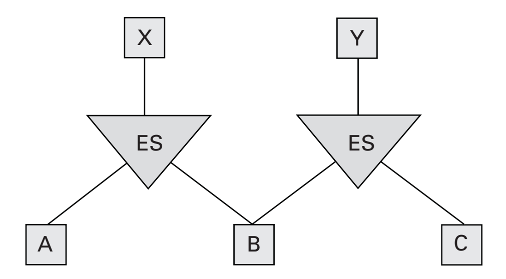


FIGURA 2.31. Diagrama E-R para el Ejercicio 2.18 (no se muestran los atributos).

2.25. Dibújense equivalentes UML de los diagramas E-R de las Figuras 2.9c, 2.10, 2.12, 2.13 y 2.17.

2.26. Considérense dos bancos separados que deciden fusionarse. Asúmase que ambos bancos usan exactamente el mismo esquema de bases de datos E-R, el de la Figura 2.22. (Obviamente, esta suposición es muy irreal; se considera un caso más realista en el Apartado 19.8.) Si la fusión del banco tiene una única base de datos, hay varios problemas potenciales:

- La posibilidad de que los dos bancos originales tengan sucursales con el mismo nombre.
- La posibilidad de que algunos clientes sean clientes de ambos bancos originales.
- La posibilidad de que algunos números de préstamo o de cuenta fueran usados en ambos bancos originales (para diferentes préstamos o cuentas, por supuesto).

Para cada uno de estos problemas potenciales descríbase por qué existen de hecho dificultades potenciales. Propóngase una solución a este problema. Explíquese cualquier cambio que se tendría que hacer para la solución y descríbase cómo afecta al esquema y a los datos.

2.27. Reconsidérese la situación descrita en el Ejercicio 2.26 bajo la suposición de que un banco está en España y el otro en Portugal. Por lo tanto, los bancos usan el esquema de la Figura 2.22, excepto que el banco portugués usa un número de identificación asignado por el gobierno portugués, mientras que el banco español usa el D.N.I. español para la identificación de clientes. ¿Qué problemas (además de los identificados en el Ejercicio 2.24) ocurrirían en este caso multinacional? ¿Cómo se podrían resolver? Asegúrese de considerar ambos esquemas y los valores de los datos actuales en la construcción de la respuesta.

NOTAS BIBLIOGRÁFICAS

El modelo de datos E-R fue introducido por Chen [1976]. Teorey et al. [1986] usó una metodología de diseño lógico para bases de datos relacionales que usa el modelo E-R extendido. La correspondencia entre los modelos E-R extendido y relacional se discutió por Lyngbaek y Vianu [1987], y por Marlowitz y Shoshani [1992]. Se han propuesto varios lenguajes de manipulación de datos para el modelo E-R: GERM (Benneworth et al. [1981]), GORDAS (Elmasri y Wiederhold [1981]) y ERROL (Markowitz y Raz [1983]). Un lenguaje de consulta gráfico para las bases de datos E-R se propuso por Zhang y Mendelzon [1983], y por ElMasri y Larson [1985].

Los conceptos de generalización, especialización y agregación se introdujeron por Smith y Smith [1977], y se extendieron en Hammer y McLeod [1981]. Lenzerini y Santucci [1983] han usado estos conceptos para definir las restricciones del modelo E-R.

Thalheim [2000] proporciona un tratamiento detallado de libros de texto de la investigación en el modelado E-R. En Batini et al. [1992] y Elmasri y Navathe [2000] se ofrecen discusiones sobre libros de texto básicos. Davis et al. [1983] proporciona una colección de artículos del modelo E-R.

HERRAMIENTAS

Muchos sistemas de bases de datos proporcionan herramientas para el diseño de bases de datos que soportan diagramas E-R. Estas herramientas ayudan al diseñador a crear diagramas E-R y pueden crear automáticamente las tablas correspondientes en una base de datos. Véanse las notas bibliográficas del Capítulo 1 para consultar referencias a sitios Web de fabricantes de bases de datos. Hay también algunas herramientas de modelado de datos independientes que soportan diagramas E-R y diagramas de clase UML. Entre ellas están Rational Rose (www.rational.com/products/rose). Visio Enterprise (véase www.visio.com) y ERwin (búsquese ERwin en el sitio www.cai.com/products).

CAPÍTULO

3 EL MODELO RELACIONAL

El modelo relacional se ha establecido actualmente como el principal modelo de datos para las aplicaciones de procesamiento de datos. Ha conseguido la posición principal debido a su simplicidad, que facilita el trabajo del programador en comparación con otros modelos anteriores como el de red y el jerárquico.

En este capítulo se estudia en primer lugar los fundamentos del modelo relacional, que proporciona una forma muy simple y potente de representar datos. A continuación se describen tres lenguajes formales de consulta; los lenguajes de consulta se usan para especificar las solicitudes de información. Los tres que se estudian en este capítulo no son cómodos de usar, pero a cambio sirven como base formal para lenguajes de consulta que sí lo son y que se estudiarán más adelante. El primer lenguaje de consulta, el álgebra relacional, se estudia en detalle. El álgebra relacional forma la base del lenguaje de consulta SQL ampliamente usado. A continuación se proporcionan visiones generales de otros dos lenguajes formales: el cálculo relacional de tuplas y el cálculo relacional de dominios, que son lenguajes declarativos de consulta basados en la lógica matemática. El cálculo relacional de dominios es la base del lenguaje QBE.

Existe una amplia base teórica de las bases de datos relacionales. En este capítulo se estudia la base teórica referida a las consultas. En el Capítulo 7 se examinarán aspectos de la teoría de bases de datos relacionales que ayudan en el diseño de esquemas de bases de datos relacionales, mientras que en los Capítulos 13 y 14 se estudian aspectos de la teoría que se refieren al procesamiento eficiente de consultas.

3.1. LA ESTRUCTURA DE LAS BASES DE DATOS RELACIONALES

Una base de datos relacional consiste en un conjunto de **tablas**, a cada una de las cuales se le asigna un nombre exclusivo. Cada tabla tiene una estructura parecida a la presentada en el Capítulo 2, donde se representaron las bases de datos E-R mediante tablas. Cada fila de la tabla representa una *relación* entre un conjunto de valores. Dado que cada tabla es un conjunto de dichas relaciones, hay una fuerte correspondencia entre el concepto de *tabla* y el concepto matemático de *relación*, del que toma su nombre el modelo de datos relacional. A continuación se introduce el concepto de relación.

En este capítulo se utilizarán varias relaciones diferentes para ilustrar los conceptos subyacentes al modelo de datos relacional. Estas relaciones representan parte de una entidad bancaria. Se diferencian ligeramente de las tablas que se utilizaron en el Capítulo 2, por lo que se puede simplificar la representación. En el Capítulo 7 se estudiarán los criterios sobre la adecuación de las estructuras relacionales.

3.1.1. Estructura básica

Considérese la tabla *cuenta* de la Figura 3.1. Tiene tres cabeceras de columna: *número-cuenta*, *nombre-sucursal* y *saldo*. Siguiendo la terminología del modelo relacional se puede hacer referencia a estas cabeceras como **atributos** (igual que se hizo en el modelo E-R en el Capítulo 2). Para cada atributo hay un conjunto de valores permitidos, llamado **dominio** de ese atributo. Para el atributo *nombre-sucursal*, por ejemplo, el dominio es el conjunto de los nombres de las sucursales. Supongamos que D_1 denota el conjunto de todos los números de cuenta, D_2 el conjunto de todos los nombres de sucursal y D_3 el conjunto de los saldos. Como se vio en el Capítulo 2 todas las filas de *cuenta* deben consistir en una tupla (v_1, v_2, v_3), donde v_1 es un número de cuenta (es decir, v_1 está en el dominio D_1), v_2 es un nombre de sucursal (es decir, v_2 está en el dominio D_2) y v_3 es un saldo (es decir, v_3 está en el dominio D_3). En general,

número-cuenta	*nombre-sucursal*	*saldo*
C-101	Centro	500
C-102	Navacerrada	400
C-201	Galapagar	900
C-215	Becerril	700
C-217	Galapagar	750
C-222	Moralzarzal	700
C-305	Collado Mediano	350

FIGURA 3.1. La relación *cuenta*.

cuenta sólo contendrá un subconjunto del conjunto de todas las filas posibles. Por tanto, *cuenta* es un subconjunto de

$$D_1 \times D_2 \times D_3$$

En general, una **tabla** de n atributos debe ser un subconjunto de

$$D_1 \times D_2 \times \ldots \times D_{n-1} \times D_n$$

Los matemáticos definen las **relaciones** como subconjuntos del producto cartesiano de la lista de dominios. Esta definición se corresponde de manera casi exacta con la definición de tabla dada anteriormente. La única diferencia es que aquí se han asignado nombres a los atributos, mientras que los matemáticos sólo utilizan «nombres» numéricos, utilizando el entero 1 para denotar el atributo cuyo dominio aparece en primer lugar en la lista de dominios, 2 para el atributo cuyo dominio aparece en segundo lugar, etcétera. Como las tablas son esencialmente relaciones, se utilizarán los términos matemáticos **relación** y **tupla** en lugar de los términos **tabla** y **fila**. Una **variable tupla** es una variable que representa a una tupla; en otras palabras, una tupla que representa al conjunto de todas las tuplas.

En la relación *cuenta* de la Figura 3.1 hay siete tuplas. Supóngase que la variable tupla t hace referencia a la primera tupla de la relación. Se utiliza la notación t[*número-cuenta*] para denotar el valor de t en el atributo *número-cuenta*. Por tanto, t[*número-cuenta*] = «C-101» y t[*nombre-sucursal*] = «Centro». De manera alternativa, se puede escribir $t[1]$ para denotar el valor de la tupla t en el primer atributo (*número-cuenta*), $t[2]$ para denotar *nombre-sucursal*, etcétera. Dado que las relaciones son conjuntos se utiliza la notación matemática $t \in r$ para denotar que la tupla t está en la relación r.

El orden en que aparecen las tuplas es irrelevante, dado que una relación es un *conjunto* de tuplas. Así, si las tuplas de una relación se muestran ordenadas como en la Figura 3.1, o desordenadas, como en la Figura 3.2, no importa; las relaciones de estas figuras son las mismas, ya que ambas contienen el mismo conjunto de tuplas.

Se exigirá que, para todas las relaciones r, los dominios de todos los atributos de r sean atómicos. Un dominio es **atómico** si los elementos del dominio se consideran unidades indivisibles. Por ejemplo, el conjunto de los enteros es un dominio atómico, pero el conjunto de todos los conjuntos de enteros es un dominio no atómico. La diferencia es que no se suele considerar que los enteros tengan subpartes, pero sí se considera que los conjuntos de enteros las tienen; por ejemplo, los enteros que forman cada conjunto. Lo importante no es lo que sea el propio dominio, sino la manera en que se utilizan los elementos del dominio en la base de datos. El dominio de todos los enteros sería no atómico si se considerase que cada entero fuera una lista ordenada de cifras. En todos los ejemplos se supondrá que los dominios son atómicos. En el Capítulo 9 se estudiarán extensiones al modelo de datos relacional para permitir dominios no atómicos.

número-cuenta	*nombre-sucursal*	*saldo*
C-101	Centro	500
C-215	Becerril	700
C-102	Navacerrada	400
C-305	Collado Mediano	350
C-201	Galapagar	900
C-222	Moralzarzal	700
C-217	Galapagar	750

FIGURA 3.2. La relación *cuenta* con las tuplas desordenadas.

Es posible que varios atributos tengan el mismo dominio. Por ejemplo, supóngase que se tiene una relación *cliente* que tiene los tres atributos *nombre-cliente*, *calle-cliente* y *ciudad-cliente* y una relación *empleado* que incluye el atributo *nombre-empleado*. Es posible que los atributos *nombre-cliente* y *nombre-empleado* tengan el mismo dominio, el conjunto de todos los nombres de personas, que en el nivel físico son cadenas de caracteres. Los dominios de *saldo* y *nombre-sucursal*, por otra parte, deberían ser distintos. Quizás es menos claro si *nombre-cliente* y *nombre-sucursal* deberían tener el mismo dominio. En el nivel físico, tanto los nombres de clientes como los nombres de sucursales son cadenas de caracteres. Sin embargo, en el nivel lógico puede que se desee que *nombre-cliente* y *nombre-sucursal* tengan dominios diferentes.

Un valor de dominio que es miembro de todos los dominios posibles es el valor **nulo**, que indica que el valor es desconocido o no existe. Por ejemplo, supóngase que se incluye el atributo *número-teléfono* en la relación *cliente*. Puede ocurrir que un cliente no tenga número de teléfono, o que su número de teléfono no figure en la guía. Entonces habrá que recurrir a los valores nulos para indicar que el valor es desconocido o que no existe. Más adelante se verá que los valores nulos crean algunas dificultades cuando se tiene acceso a la base de datos o cuando se actualiza y que, por tanto, deben eliminarse si es posible. Se asumirá inicialmente que no hay valores nulos y en el Apartado 3.3.4 se describirá el efecto de los valores nulos en las diferentes operaciones.

3.1.2. Esquema de la base de datos

Cuando se habla de bases de datos se debe diferenciar entre el **esquema de la base de datos**, o diseño lógico de la misma, y el **ejemplar de la base de datos**, que es una instantánea de los datos de la misma en un momento dado.

El concepto de relación se corresponde con el concepto de variable de los lenguajes de programación. El concepto de **esquema de la relación** se corresponde con el concepto de definición de tipos de los lenguajes de programación.

Resulta conveniente dar un nombre a los esquemas de las relaciones, igual que se dan nombres a las definiciones de tipos en los lenguajes de programación. Se adopta el convenio de utilizar nombres en minúsculas para las relaciones y nombres que comiencen por una letra mayúscula para los esquemas de las relaciones. Siguiendo esta notación se utilizará *Esquema-cuenta* para denotar el esquema de la relación de la relación *cuenta*. Por tanto,

Esquema-cuenta = (*número-cuenta, nombre-sucursal, saldo*)

Se denota el hecho de que *cuenta* es una relación de *Esquema-cuenta* mediante

cuenta (*Esquema-cuenta*)

En general, los esquemas de las relaciones incluyen una lista de los atributos y de sus dominios correspondientes. La definición exacta del dominio de cada atributo no será relevante hasta que se discuta el lenguaje SQL en el Capítulo 4.

El concepto de **ejemplar de relación** se corresponde con el concepto de valor de una variable en los lenguajes de programación. El valor de una variable dada puede cambiar con el tiempo; de manera parecida, el contenido del ejemplar de una relación puede cambiar con el tiempo cuando la relación se actualiza. Sin embargo, se suele decir simplemente «relación» cuando realmente se quiere decir «ejemplar de la relación».

Como ejemplo de ejemplar de una relación, considérese la relación *sucursal* de la Figura 3.3. El esquema de esa relación es

Esquema-relación = (*nombre-sucursal, ciudad-sucursal, activos*)

Obsérvese que el atributo *nombre de la sucursal* aparece tanto en *Esquema-sucursal* como en *Esquema-cuenta*. Esta duplicidad no es una coincidencia. Más bien, utilizar atributos comunes en los esquemas de las relaciones es una manera de relacionar las tuplas de relaciones diferentes. Por ejemplo, supóngase que se desea obtener información sobre todas las cuentas abiertas en sucursales ubicadas en Arganzuela. Primero se busca en la relación *sucursal* para encontrar los nombres de todas las sucursales sitas en Arganzuela. Luego, para cada una de ellas, se mira en la relación *cuenta* para encontrar la información sobre las cuentas abiertas en esa sucursal. Esto no es sorprendente: recuérdese que los atributos que forma la clave primaria de un conjunto de entidades fuertes aparecen en la tabla creada para representar el conjunto de entidades, así como en las tablas creadas para crear relaciones en las que participar el conjunto de entidades.

Continuemos con el ejemplo bancario. Se necesita una relación que describa la información sobre los clientes. El esquema de la relación es:

Esquema-cliente = (*nombre-cliente, calle-cliente, ciudad-cliente*)

En la Figura 3.4 se muestra un ejemplo de la relación *cliente* (*Esquema-cliente*). Obsérvese que se ha omitido el atributo *id-cliente*, que se usó en el Capítulo 2, porque no se desea tener esquemas de relación más pequeños en este ejemplo. Se asume que el nombre de cliente identifica unívocamente un cliente; obviamente, esto no es cierto en el mundo real, pero las suposiciones hechas en estos ejemplos los hacen más sencillos de entender.

En una base de datos del mundo real, *id-cliente* (que podría ser el número de la seguridad social o un identificador generado por el banco) serviría para identificar unívocamente a los clientes.

También se necesita una relación que describa la asociación entre los clientes y las cuentas. El esquema de la relación que describe esta asociación es:

Esquema-impositor = (*nombre-cliente, número-cuenta*)

En la Figura 3.5 se muestra un ejemplo de la relación *impositor* (*Esquema-impositor*).

Puede parecer que, para el presente ejemplo bancario, se podría tener sólo un esquema de relación, en vez de tener varios. Es decir, puede resultar más sencillo para el usuario pensar en términos de un esquema de

nombre de la sucursal	*ciudad de la sucursal*	*activos*
Galapagar	Arganzuela	7.500
Centro	Arganzuela	9.000.000
Becerril	Aluche	2.000
Segovia	Cerceda	3.700.000
Navacerrada	Aluche	1.700.000
Navas de la Asunción	Alcalá de Henares	1.500
Moralzarzal	La Granja	2.500
Collado Mediano	Aluche	8.000.000

FIGURA 3.3. La relación *sucursal*.

nombre-cliente	*calle-cliente*	*ciudad-cliente*
Abril	Preciados	Valsaín
Amo	Embajadores	Arganzuela
Badorrey	Delicias	Valsaín
Fernández	Jazmín	León
Gómez	Carretas	Cerceda
González	Arenal	La Granja
López	Mayor	Peguerinos
Pérez	Carretas	Cerceda
Rodríguez	Yeserías	Cádiz
Rupérez	Ramblas	León
Santos	Mayor	Peguerinos
Valdivieso	Goya	Vigo

FIGURA 3.4. La relación *cliente*.

nombre cliente	*número cuenta*
Abril	C-102
Gómez	C-101
González	C-201
González	C-217
López	C-222
Rupérez	C-215
Santos	C-305

FIGURA 3.5. La relación *impositor*.

número-préstamo	*nombre-sucursal*	*importe*
P-11	Collado Mediano	900
P-14	Centro	1.500
P-15	Navacerrada	1.500
P-16	Navacerrada	1.300
P-17	Centro	1.000
P-23	Moralzarzal	2.000
P-93	Becerril	500

FIGURA 3.6. La relación *préstamo*.

relación, en lugar de en términos de varios esquemas. Supóngase que sólo se utilizara una relación para el ejemplo, con el esquema

(*nombre-sucursal, ciudad-sucursal, activos, nombre-cliente, calle-cliente, ciudad-cliente, número-cuenta, saldo*)

Obsérvese que si un cliente tiene varias cuentas hay que repetir su dirección una vez por cada cuenta. Es decir, hay que repetir varias veces parte de la información. Esta repetición supone un gasto inútil y se evita mediante el uso de varias relaciones, como en el ejemplo presente.

Además, si una sucursal no tiene ninguna cuenta (por ejemplo, una sucursal recién creada que todavía no tiene clientes), no se puede construir una tupla completa en la relación única anterior, dado que no hay todavía ningún dato disponible referente a *cliente* ni a *cuenta*. Para representar las tuplas incompletas hay que utilizar valores *nulos* que indiquen que ese valor es desconocido o no existe. Por tanto, en el ejemplo presente, los valores de *nombre-cliente*, *calle-cliente*, etcétera, deben quedar nulos. Utilizando varias relaciones se puede representar la información de las sucursales de un banco sin clientes sin utilizar valores nulos. Sencillamente, se utiliza una tupla en *Esquema-sucursal* para representar la información de la sucursal y sólo crear tuplas en los otros esquemas cuando esté disponible la información adecuada.

En el Capítulo 7 se estudiarán los criterios para decidir cuándo un conjunto de esquemas de relaciones es más apropiado que otro en términos de repetición de la información y de la existencia de valores nulos. Por ahora se supondrá que los esquemas de las relaciones vienen dados de antemano.

Se incluyen dos relaciones más para describir los datos de los préstamos concedidos en las diferentes sucursales del banco:

Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
Esquema-prestatario = (*nombre-cliente, número-préstamo*)

Las relaciones de ejemplo *préstamo* (*Esquema-préstamo*) y *prestatario* (*Esquema-prestatario*) se muestran en las Figuras 3.5 y 3.6, respectivamente.

La entidad bancaria que se ha descrito se deriva del diagrama E-R mostrado en la Figura 3.8. Los esquemas de las relaciones se corresponden con el conjunto de tablas que se podrían generar utilizando el método esbozado en el Apartado 2.9. Obsérvese que las tablas para *cuenta-sucursal* y *préstamo-sucursal* se han combinado en las tablas de *cuenta* y *préstamo* respectivamente. Esta combinación es posible dado que las relaciones son de varios a uno desde *cuenta* y *préstamo*, respectivamente, a *sucursal* y, además, la participación de *cuenta* y *préstamo* en las relaciones correspondientes es total, como indican las líneas dobles en la figura. Finalmente, obsérvese que la relación *cliente* puede contener información sobre clientes que ni tengan cuenta ni un préstamo en el banco.

La entidad bancaria aquí descrita servirá como ejemplo principal en este capítulo y en los siguientes. Cuando sea necesario, habrá que introducir más esquemas de relaciones para ilustrar casos concretos.

3.1.3. Claves

Los conceptos de *superclave*, de *clave candidata* y de *clave primaria*, tal y como se discute en el Capítulo 2, también son aplicables en el modelo relacional. Por ejemplo, en *Esquema-sucursal*, tanto {*nombre-sucursal*} como {*nombre-sucursal*, *ciudad-sucursal*} son superclaves. {*nombre-sucursal*, *ciudad-sucursal*} no es una clave candidata porque {*nombre-sucursal*} es un subconjunto de {*nombre-sucursal*, *ciudad-sucursal*} y {*nombre-sucursal*} es una superclave. Sin embargo, {*nombre-sucursal*} es una clave candidata, y servirá también como clave primaria para estos fines. El atributo *ciudad-sucursal* no es una superclave, dado que dos sucursales de la misma ciudad pueden tener nombres diferentes (y diferentes volúmenes de activos).

nombre cliente	*número préstamo*
Fernández	P-16
Gómez	P-93
Gómez	P-15
López	P-14
Pérez	P-17
Santos	P-11
Sotoca	P-23
Valdivieso	P-17

FIGURA 3.7. La relación *prestatario*.

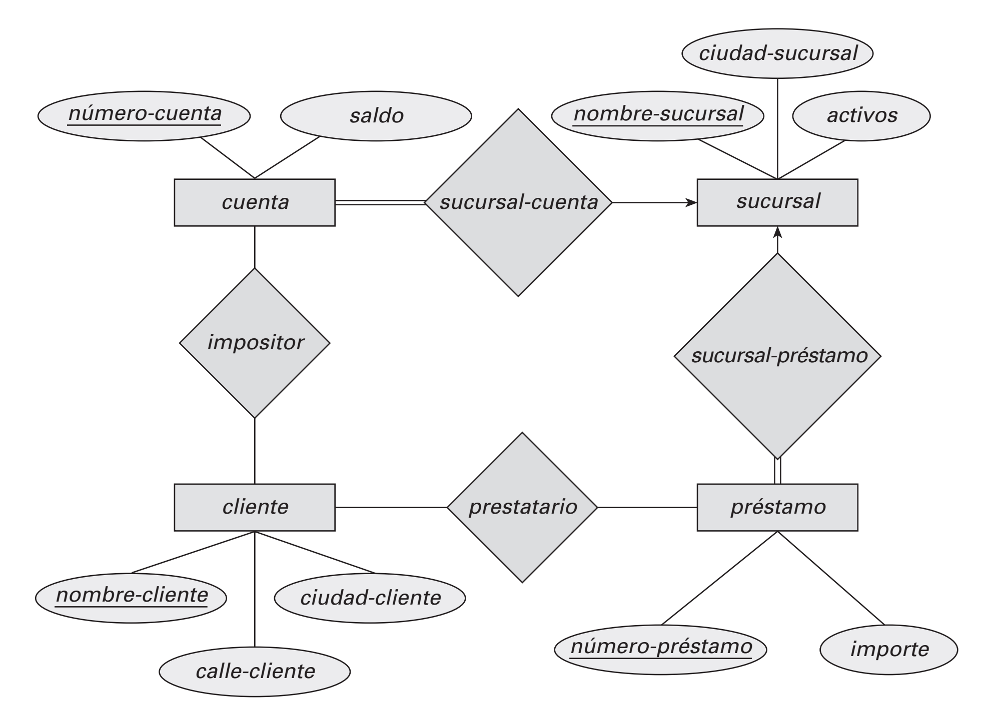


FIGURA 3.8. Diagrama E-R de la entidad bancaria.

Sea R el esquema de una relación. Si se dice que un subconjunto K de R es una *superclave* de R para las relaciones $r(R)$ en las que no hay dos tuplas diferentes que tengan los mismos valores en todos los atributos de K. Es decir, si t_1 y t_2 están en R y $t_1 \neq t_2$, entonces $t_1[K] \neq t_2[K]$.

Si el esquema de una base de datos relacional se basa en las tablas derivadas de un esquema E-R es posible determinar la clave primaria del esquema de una relación a partir de las claves primarias de los conjuntos de entidades o de relaciones de los que se deriva el esquema:

- **Conjunto de entidades fuertes.** La clave primaria del conjunto de entidades se convierte en la clave primaria de la relación.
- **Conjunto de entidades débiles.** La tabla y, por tanto, la relación correspondientes a un conjunto de entidades débiles incluyen
  - — Los atributos del conjunto de entidades débiles.
  - — La clave primaria del conjunto de entidades fuertes del que depende el conjunto de entidades débiles.

  La clave primaria de la relación consiste en la unión de la clave primaria del conjunto de entidades fuertes y el discriminante del conjunto de entidades débil.
- **Conjunto de relaciones.** La unión de las claves primarias de los conjuntos de entidades relacionados se transforma en una superclave de la relación. Si la relación es de varios a varios, esta superclave es también la clave primaria. En el Apartado 2.4.2 se describe la manera de determinar las claves primarias en otros casos. Recuérdese del Apartado 2.9.3 que no se genera ninguna tabla para los conjuntos de relaciones que vinculan un conjunto de entidades débiles con el conjunto de entidades fuertes correspondiente.
- **Tablas combinadas.** Recuérdese del Apartado 2.9.3 que un conjunto binario de relaciones de varios a uno entre A y B puede representarse mediante una tabla que consista en los atributos de A y en los atributos (si hay alguno) del conjunto de relaciones. La clave primaria de la entidad «varios» se transforma en la clave primaria de la relación (es decir, si el conjunto de relaciones es de varios a uno entre A y B, la clave primaria de A es la clave primaria de la relación). Para los conjuntos de relaciones de uno a uno la relación se construye igual que en el conjunto de relaciones de varios a uno. Sin embargo, cualquiera de las claves primarias del conjunto de entidades puede elegirse como clave primaria de la relación, dado que ambas son claves candidatas.
- **Atributos multivalorados.** Recuérdese del Apartado 2.9.4 que un atributo multivalorado M se representa mediante una tabla consistente en la clave primaria del conjunto de entidades o de relaciones del que M es atributo y en una columna C que guarda un valor concreto de M. La clave primaria del conjunto de entidades o de relaciones

junto con el atributo *C* se convierte en la clave primaria de la relación.

A partir de la lista precedente se puede ver que el esquema de una relación puede incluir entre sus atributos la clave primaria de otro esquema, digamos r_2. Este atributo es una **clave externa** de r_1 que hace referencia a r_2. La relación r_1 también se denomina la **relación referenciante** de la dependencia de clave externa, y r_2 se denomina la **relación referenciada** de la clave externa. Por ejemplo, el atributo *nombre-sucursal* de *Esquema-cuenta* es una clave externa de *Esquema-cuenta* referenciando a *Esquema-sucursal*, ya que *nombre-sucursal* es la clave primaria de *Esquema-sucursal*. En cualquier ejemplar de la base de datos, dada una tupla t_a de la relación *cuenta*, debe haber alguna tupla t_b en la relación *cuenta* tal que el valor del atributo *nombre-sucursal* de t_a sea el mismo que el valor de la clave primaria, *nombre-sucursal*, de t_b.

Es obligado listar los atributos que forman clave primaria de un esquema de relación antes que el resto; por ejemplo, el atributo *nombre-sucursal* de *Esquema-sucursal* se lista en primer lugar, ya que es la clave primaria.

3.1.4. Diagramas de esquema

Un esquema de bases de datos, junto con las dependencias de clave primaria y externa, se puede mostrar gráficamente mediante **diagramas de esquema**. La Figura 3.9 muestra el diagrama de esquema del ejemplo bancario. Cada relación aparece como un cuadro con los atributos listados dentro de él y el nombre de la relación sobre él. Si hay atributos clave primaria, una línea horizontal cruza el cuadro con los atributos clave primaria listados sobre ella. Las dependencias de clave externa aparecen como flechas desde los atributos clave externa de la relación referenciante a la clave primaria de la relación referenciada.

No hay que confundir un diagrama de esquema con un diagrama E-R. En particular, los diagramas E-R no muestran explícitamente los atributos clave externa, mientras que los diagramas de esquema sí.

Muchos sistemas de bases de datos proporcionan herramientas de diseño con una interfaz gráfica de usuario para la creación de diagramas de esquema.

3.1.5. Lenguajes de consulta

Un **lenguaje de consulta** es un lenguaje en el que un usuario solicita información de la base de datos. Estos lenguajes suelen ser de un nivel superior que el de los lenguajes de programación habituales. Los lenguajes de consulta pueden clasificarse como procedimentales o no procedimentales. En los **lenguajes procedimentales** el usuario instruye al sistema para que lleve a cabo una serie de operaciones en la base de datos para calcular el resultado deseado. En los **lenguajes no procedimentales** el usuario describe la información deseada sin dar un procedimiento concreto para obtener esa información.

La mayor parte de los sistemas comerciales de bases de datos relacionales ofrecen un lenguaje de consulta que incluye elementos de los enfoques procedimental y no procedimental. Se estudiarán varios lenguajes comerciales en el Capítulo 4. El Capítulo 5 trata los lenguajes QBE y Datalog, este último parecido a Prolog.

En este capítulo se examinarán los lenguajes «puros»: el álgebra relacional es procedimental, mientras que el cálculo relacional de tuplas y el de dominios son no procedimentales. Estos lenguajes de consulta son rígidos y formales, y carecen del «azúcar sintáctico» de los lenguajes comerciales, pero ilustran las técnicas fundamentales para la extracción de datos de las bases de datos.

Aunque inicialmente sólo se estudiarán las consultas, un lenguaje de manipulación de datos completo no sólo incluye un lenguaje de consulta, sino también un lenguaje para la modificación de las bases de datos. Estos lenguajes incluyen órdenes para insertar y borrar tuplas, así como órdenes para modificar partes de las tuplas existentes. Las modificaciones de las bases de datos se examinarán después de completar la discusión sobre las consultas.

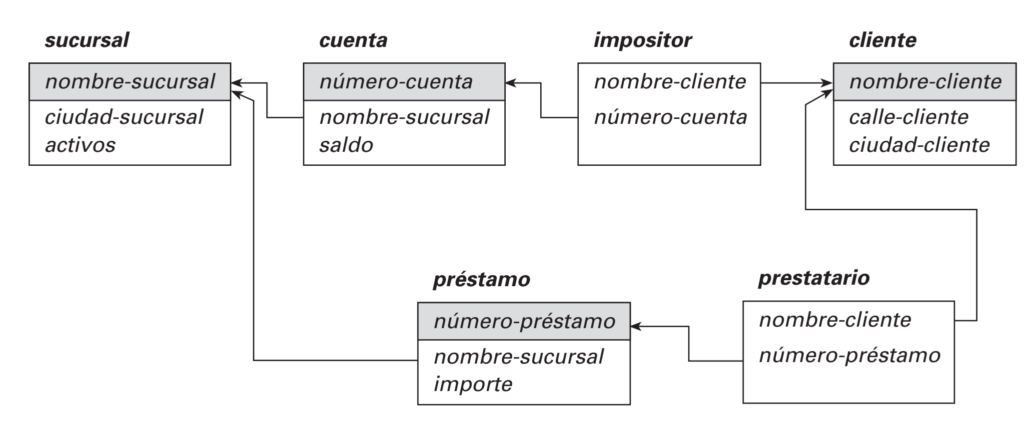


FIGURA 3.9. Diagrama de esquema para el banco.

3.2. EL ÁLGEBRA RELACIONAL

El álgebra relacional es un lenguaje de consulta *procedimental*. Consta de un conjunto de operaciones que toman como entrada una o dos relaciones y producen como resultado una nueva relación. Las operaciones fundamentales del álgebra relacional son *selección*, *proyección*, *unión*, *diferencia de conjuntos*, *producto cartesiano* y *renombramiento*. Además de las operaciones fundamentales hay otras operaciones, por ejemplo, intersección de conjuntos, reunión natural, división y asignación. Estas operaciones se definirán en términos de las operaciones fundamentales.

3.2.1. Operaciones fundamentales

Las operaciones selección, proyección y renombramiento se denominan operaciones *unarias* porque operan sobre una sola relación. Las otras tres operaciones operan sobre pares de relaciones y se denominan, por lo tanto, operaciones *binarias*.

3.2.1.1. La operación selección

La operación **selección** selecciona tuplas que satisfacen un predicado dado. Se utiliza la letra griega sigma minúscula (σ) para denotar la selección. El predicado aparece como subíndice de σ. La relación del argumento se da entre paréntesis a continuación de σ. Por tanto, para seleccionar las tuplas de la relación *préstamo* en que la sucursal es «Navacerrada» hay que escribir

$$\sigma_{nombre\text{-}sucursal\ =\ \text{«Navacerrada»}}\ (préstamo)$$

Si la relación *préstamo* es como se muestra en la Figura 3.6, la relación que resulta de la consulta anterior es como se muestra en la Figura 3.10.

Se pueden buscar todas las tuplas en las que el importe prestado sea mayor que 1.200 € escribiendo

$$\sigma_{importe>1200}\ (préstamo)$$

En general, se permiten las comparaciones que utilizan =, ≠, <, ≤, > o ≥ en el predicado de selección. Además, se pueden combinar varios predicados en uno mayor utilizando las conectivas *y* (∧) y *o* (∨). Por tanto, para encontrar las tuplas correspondientes a préstamos de más de 1.200 € concedidos por la sucursal de Navacerrada, se escribe

$$\sigma_{nombre\text{-}sucursal\ =\ \text{«Navacerrada»}\ \wedge\ importe>1200}\ (préstamo)$$

número-préstamo	*nombre-sucursal*	*importe*
P-15	Navacerrada	1.500
P-16	Navacerrada	1.300

FIGURA 3.10. Resultado de $\sigma_{\text{nombre-sucursal = «Navacerrada»}}$ (préstamo).

El predicado de selección puede incluir comparaciones entre dos atributos. Para ilustrarlo, considérese la relación *responsable-préstamo*, que consta de tres atributos: *nombre-cliente*, *nombre-banquero* y *número-préstamo*, que especifica que un empleado concreto es el responsable del préstamo concedido a un cliente. Para hallar todos los clientes que se llaman igual que su responsable de préstamos se puede escribir

$$\sigma_{nombre\text{-}cliente\ =\ nombre\text{-}banquero}\ (responsable\text{-}préstamo)$$

Dado que el valor especial *nulo* indica «valor desconocido o inexistente», cualquier comparación que implique a un valor nulo se evalúa como **falsa**.

3.2.1.2. La operación proyección

Supóngase que se desea hacer una lista de todos los números de préstamo y del importe de los mismos, pero sin que aparezcan los nombres de las sucursales. La operación **proyección** permite producir esta relación. La operación proyección es una operación unaria que devuelve su relación de argumentos, excluyendo algunos argumentos. Dado que las relaciones son conjuntos, se eliminan todas las filas duplicadas. La proyección se denota por la letra griega mayúscula pi (Π). Se crea una lista de los atributos que se desea que aparezcan en el resultado como subíndice de Π. La relación de argumentos se escribe a continuación entre paréntesis. Por tanto, la consulta para crear una lista de todos los números de préstamo y del importe de los mismos puede escribirse como

$$\Pi_{número\text{-}préstamo,\ importe}\ (préstamo)$$

La relación que resulta de esta consulta se muestra en la Figura 3.11.

3.2.1.3. Composición de operaciones relacionales

Es importante el hecho de que el resultado de una operación relacional sea también una relación. Considérese la consulta más compleja «Encontrar los clientes que viven en Peguerinos». Hay que escribir:

número-préstamo	*importe*
P-11	900
P-14	1.500
P-15	1.500
P-16	1.300
P-17	1.000
P-23	2.000
P-93	500

FIGURA 3.11. Números de préstamo y sus importes.

$$\Pi_{nombre\text{-}cliente}\ (\sigma_{ciudad\text{-}cliente\ =\ «\text{Peguerinos}»}\ (cliente))$$

Téngase en cuenta que, en vez de dar en el argumento de la operación proyección el nombre de una relación, se da una expresión que se evalúa como una relación.

En general, dado que el resultado de una operación del álgebra relacional es del mismo tipo (relación) que los datos de entrada, las operaciones del álgebra relacional pueden componerse para formar una **expresión del álgebra relacional**. La composición de operaciones del álgebra relacional para formar expresiones del álgebra relacional es igual que la composición de operaciones aritméticas (como +, –, ∗ y ÷) para formar expresiones aritméticas. La definición formal de las expresiones de álgebra relacional se estudia en el Apartado 3.2.2.

3.2.1.4. La operación unión

Considérese una consulta para averiguar el nombre de todos los clientes del banco que tienen una cuenta, un préstamo o ambas cosas. Obsérvese que la relación *cliente* no contiene esa información, dado que los clientes no necesitan tener ni cuenta ni préstamo en el banco. Para contestar a esta consulta hace falta la información de la relación *impositor* (Figura 3.5) y la de la relación *prestatario* (Figura 3.7). Se conoce la manera de averiguar los nombres de todos los clientes con préstamos en el banco:

$$\Pi_{nombre\text{-}cliente}\ (prestatario)$$

También se conoce la manera de averiguar el nombre de los clientes con cuenta en el banco:

$$\Pi_{nombre\text{-}cliente}\ (impositor)$$

Para contestar a la consulta hace falta la **unión** de estos dos conjuntos; es decir, hacen falta todos los nombres de clientes que aparecen en alguna de las dos relaciones o en ambas. Estos datos se pueden averiguar mediante la operación binaria unión, denotada, como en la teoría de conjuntos, por ∪. Por tanto, la expresión buscada es

$$\Pi_{nombre\text{-}cliente}\ (prestatario) \cup \Pi_{nombre\text{-}cliente}\ (impositor)$$

La relación resultante de esta consulta aparece en la Figura 3.10. Téngase en cuenta que en el resultado hay diez tuplas, aunque hay siete prestatarios y seis impositores distintos. Esta discrepancia aparente se debe a que Gómez, Santos y López son a la vez prestatarios e impositores. Dado que las relaciones son conjuntos, se eliminan los valores duplicados.

Obsérvese que en este ejemplo se toma la unión de dos conjuntos, ambos consistentes en valores de *nombre-cliente*. En general, se debe asegurar que las uniones se realicen entre relaciones *compatibles*. Por ejemplo, no tendría sentido realizar la unión de las relaciones *préstamo* y *prestatario*. La primera es una relación con tres atributos, la segunda sólo tiene dos. Más aún, considérese la unión de un conjunto de nombres de clientes y de un conjunto de ciudades. Una unión así no tendría sentido en la mayor parte de los casos. Por tanto, para que una operación unión $r \cup s$ sea válida hay que exigir que se cumplan dos condiciones:

1. Las relaciones *r* y *s* deben ser de la misma aridad. Es decir, deben tener el mismo número de atributos.
2. Los dominios de los atributos *i*-ésimos de *r* y de *s* deben ser iguales para todo *i*.

Téngase en cuenta que *r* y *s* pueden ser, en general, relaciones temporales que sean resultado de expresiones del álgebra relacional.

nombre-cliente
Abril
Fernández
Gómez
González
López
Pérez
Rupérez
Santos
Sotoca
Valdivieso

FIGURA 3.12. Nombres de todos los clientes que tienen un préstamo o una cuenta.

3.2.1.5. La operación diferencia de conjuntos

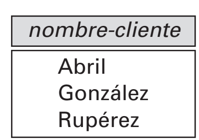
La operación **diferencia de conjuntos**, denotada por –, permite buscar las tuplas que estén en una relación pero no en la otra. La expresión $r - s$ da como resultado una relación que contiene las tuplas que están en *r* pero no en *s*.

Se pueden buscar todos los clientes del banco que tienen abierta una cuenta pero no tienen concedido ningún préstamo escribiendo

$$\Pi_{nombre\text{-}cliente}\ (impositor) - \Pi_{nombre\text{-}cliente}\ (prestatario)$$

La relación resultante de esta consulta aparece en la Figura 3.13.

Como en el caso de la operación unión, hay que asegurarse de que las diferencias de conjuntos se realicen entre relaciones *compatibles*. Por tanto, para que una



nombre-cliente
Abril
González
Rupérez

FIGURA 3.13. Clientes con cuenta abierta pero sin préstamo concedido.

operación diferencia de conjuntos $r - s$ sea válida hay que exigir que las relaciones r y s sean de la misma aridad y que los dominios de los atributos i-ésimos de r y s sean iguales.

3.2.1.6. La operación producto cartesiano

La operación **producto cartesiano**, denotada por un aspa (×), permite combinar información de cualesquiera dos relaciones. El producto cartesiano de las relaciones r_1 y r_2 como $r_1 \times r_2$.

Recuérdese que las relaciones se definen como subconjuntos del producto cartesiano de un conjunto de dominios. A partir de esta definición ya se debe tener una intuición sobre la definición de la operación producto cartesiano. Sin embargo, dado que el mismo nombre de atributo puede aparecer tanto en r_1 como en r_2, hay que crear un esquema de denominaciones para distinguir entre ambos atributos. En este caso se logra adjuntando al atributo el nombre de la relación de la que proviene originalmente. Por ejemplo, el esquema de relación de r = *prestatario* × *préstamo* es

(*prestatario.nombre-cliente, prestatario.número-préstamo, préstamo.nombre-sucursal, préstamo.número-préstamo, préstamo.importe*)

Con este esquema se puede distinguir entre *prestatario.número-préstamo* y *préstamo.número-préstamo*. Para los atributos que sólo aparecen en uno de los dos esquemas se suele omitir el prefijo con el nombre de la relación. Esta simplificación no genera ambigüedad alguna. Por tanto, se puede escribir el esquema de relación de r como

(*nombre-cliente, prestatario.número-préstamo, nombre-sucursal, préstamo.número-préstamo, importe*)

El acuerdo de denominaciones precedente *exige* que las relaciones que sean argumentos de la operación producto cartesiano tengan nombres diferentes. Esta exigencia causa problemas en algunos casos, como cuando se desea calcular el producto cartesiano de una relación consigo misma. Se produce un problema similar si se utiliza el resultado de una expresión del álgebra relacional en un producto cartesiano, dado que hará falta un nombre para la relación para poder hacer referencia a sus atributos. En el Apartado 3.2.1.7 se verá la manera de evitar estos problemas utilizando una operación renombramiento.

Ahora que se conoce el esquema de relación de r = *prestatario* × *préstamo* hay que averiguar las tuplas que aparecerán en r. Como se podía imaginar, se crea una tupla de r a partir de cada par de tuplas posible: una de la relación *prestatario* y otra de la relación *préstamo*. Por tanto, r es una relación de gran tamaño, como se puede ver en la Figura 3.14, donde sólo se ha incluido una parte de las tuplas que forman parte de r.

Supóngase que se tienen n_1 tuplas en *prestatario* y n_2 tuplas en *préstamo*. Por tanto, hay $n_1 * n_2$ maneras de escoger un par de tuplas, una tupla de cada relación; por lo que hay $n_1 * n_2$ tuplas en r. En concreto, obsérvese que para algunas tuplas t de r puede ocurrir que t[*prestatario.número-préstamo*] ≠ t[*préstamo.número-préstamo*].

En general, si se tienen las relaciones r_1 (R_1) y r_2 (R_2), $r_1 \times r_2$ es una relación cuyo esquema es la concatenación de R_1 y de R_2. La relación R contiene todas las tuplas t para las que hay unas tuplas t_1 en r_1 y t_2 en r_2 para las que $t[R_1] = t_1[R_1]$ y $t[R_2] = t_2[R_2]$.

Supóngase que se desea averiguar los nombres de todos los clientes que tienen concedido un préstamo en la sucursal de Navacerrada. Se necesita para ello información de las relaciones *préstamo* y *prestatario*. Si se escribe

$$\sigma_{\text{nombre-sucursal = «Navacerrada»}} (\textit{prestatario} \times \textit{préstamo})$$

entonces el resultado es la relación mostrada en la Figura 3.15. Se tiene una relación que sólo atañe a la sucursal de Navacerrada. Sin embargo, la columna *nombre-cliente* puede contener clientes que no tengan concedido ningún préstamo en la sucursal de Navacerrada. (Si no se ve el motivo por el que esto es cierto, recuérdese que el producto cartesiano toma todos los emparejamientos posibles de una tupla de *prestatario* con una tupla de *préstamo*.)

Dado que la operación producto cartesiano asocia *todas* las tuplas de *préstamo* con todas las tuplas de *prestatario*, se sabe que, si un cliente tiene concedido un préstamo en la sucursal de Navacerrada, hay alguna tupla de *prestatario* × *préstamo* que contiene su nombre y que *prestatario.número-préstamo* = *préstamo.número-préstamo*. Por tanto, si escribimos

$$\sigma_{\textit{prestatario.número-préstamo = préstamo.número-préstamo}} (\sigma_{\textit{nombre-sucursal} = \text{«Navacerrada»}} (\textit{prestatario} \times \textit{préstamo}))$$

sólo se obtienen las tuplas de *prestatario* × *préstamo* que corresponden a los clientes que tienen concedido un préstamo en la sucursal de Navacerrada.

Finalmente, dado que sólo se desea obtener *nombre-cliente*, se realiza una proyección:

$$\Pi_{\textit{nombre-cliente}} (\sigma_{\textit{prestatario.número-préstamo = préstamo.número-préstamo}} (\sigma_{\textit{nombre-sucursal} = \text{«Navacerrada»}} (\textit{prestatario} \times \textit{préstamo})))$$

El resultado de esta expresión se muestra en la Figura 3.16 y es la respuesta correcta a la consulta formulada.

3.2.1.7. La operación renombramiento

A diferencia de las relaciones de la base de datos, los resultados de las expresiones de álgebra relacional no tienen un nombre que se pueda utilizar para referirse a ellas. Resulta útil poder ponerles nombre; el operador

nombre-cliente	*prestatario.número-préstamo*	*préstamo.número-préstamo*	*nombre-sucursal*	*importe*
Santos	P-17	P-11	Collado Mediano	900
Santos	P-17	P-14	Centro	1.500
Santos	P-17	P-15	Navacerrada	1.500
Santos	P-17	P-16	Navacerrada	1.300
Santos	P-17	P-17	Centro	1.000
Santos	P-17	P-23	Moralzarzal	2.000
Santos	P-17	P-93	Becerril	500
Gómez	P-23	P-11	Collado Mediano	900
Gómez	P-23	P-14	Centro	1.500
Gómez	P-23	P-15	Navacerrada	1.500
Gómez	P-23	P-16	Navacerrada	1.300
Gómez	P-23	P-17	Centro	1.000
Gómez	P-23	P-23	Moralzarzal	2.000
Gómez	P-23	P-93	Becerril	500
López	P-15	P-11	Collado Mediano	900
López	P-15	P-14	Centro	1.500
López	P-15	P-15	Navacerrada	1.500
López	P-15	P-16	Navacerrada	1.300
López	P-15	P-17	Centro	1.000
López	P-15	P-23	Moralzarzal	2.000
López	P-15	P-93	Becerril	500
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…
Valdivieso	P-17	P-11	Collado Mediano	900
Valdivieso	P-17	P-14	Centro	1.500
Valdivieso	P-17	P-15	Navacerrada	1.500
Valdivieso	P-17	P-16	Navacerrada	1.300
Valdivieso	P-17	P-17	Centro	1.000
Valdivieso	P-17	P-23	Moralzarzal	2.000
Valdivieso	P-17	P-93	Becerril	500
Fernández	P-16	P-11	Collado Mediano	900
Fernández	P-16	P-14	Centro	1.500
Fernández	P-16	P-15	Navacerrada	1.500
Fernández	P-16	P-16	Navacerrada	1.300
Fernández	P-16	P-17	Centro	1.000
Fernández	P-16	P-23	Moralzarzal	2.000
Fernández	P-16	P-93	Becerril	500

FIGURA 3.14. Resultado de *prestatario* × *préstamo*.

nombre-cliente	*prestatario.número-préstamo*	*préstamo.número-préstamo*	*nombre-sucursal*	*importe*
Santos	P-17	P-15	Navacerrada	1.500
Santos	P-17	P-16	Navacerrada	1.300
Gómez	P-23	P-15	Navacerrada	1.500
Gómez	P-23	P-16	Navacerrada	1.300
López	P-15	P-15	Navacerrada	1.500
López	P-15	P-16	Navacerrada	1.300
Sotoca	P-14	P-15	Navacerrada	1.500
Sotoca	P-14	P-16	Navacerrada	1.300
Pérez	P-93	P-15	Navacerrada	1.500
Pérez	P-93	P-16	Navacerrada	1.300
Gómez	P-11	P-15	Navacerrada	1.500
Gómez	P-11	P-16	Navacerrada	1.300
Valdivieso	P-17	P-15	Navacerrada	1.500
Valdivieso	P-17	P-16	Navacerrada	1.300
Fernández	P-16	P-15	Navacerrada	1.500
Fernández	P-16	P-16	Navacerrada	1.300

FIGURA 3.15. Resultado de $\sigma_{\text{nombre-sucursal = «Navacerrada»}}$ (*prestatario* × *préstamo*).

nombre-cliente
Férnandez
López

FIGURA 3.16. Resultado de $\Pi_{nombre\text{-}cliente}$ ($\sigma_{prestatario.número\text{-}préstamo = préstamo.número\text{-}préstamo}$ ($\sigma_{nombre\text{-}sucursal = «Navacerrada»}$ (*prestatario* × *préstamo*))).

saldo
500
400
700
750
350

FIGURA 3.17. Resultado de la subexpresión $\Pi_{cuenta\text{-}saldo}$ ($\sigma_{cuenta\text{-}saldo < d.saldo}$ (*cuenta* × ρ_d (*cuenta*))).

renombramiento, denotado por la letra griega rho minúscula (ρ), permite realizar esta tarea. Dada una expresión *E* del álgebra relacional, la expresión

$$\rho_x (E)$$

devuelve el resultado de la expresión *E* con el nombre *x*.

Las relaciones *r* por sí mismas se consideran expresiones (triviales) del álgebra relacional. Por tanto, también se puede aplicar la operación renombramiento a una relación *r* para obtener la misma relación con un nombre nuevo.

Otra forma de la operación renombramiento es la siguiente. Supóngase que una expresión del álgebra relacional *E* tiene aridad *n*. Por tanto, la expresión

$$\rho_{x\,(A_1,\, A_2,\, \ldots\, A_n)} (E)$$

devuelve el resultado de la expresión *E* con el nombre *x* y con los atributos con el nombre cambiado a A_1, A_2, …, A_n.

Para ilustrar el uso del renombramiento de las relaciones, considérese la consulta «Buscar el máximo saldo de cuenta del banco». La estrategia empleada para obtener el resultado es 1) calcular una relación intermedia consistente en los saldos que *no* son el máximo y 2) realizar la diferencia entre la relación Π_{saldo} (*cuenta*) y la relación intermedia recién calculada.

Paso 1: Para calcular la relación intermedia hay que comparar los valores de los saldos de todas las cuentas. Esta comparación se puede hacer calculando el producto cartesiano *cuenta* × *cuenta* y formando una selección para comparar el valor de cualesquiera dos saldos que aparezcan en una tupla. En primer lugar hay que crear un mecanismo para distinguir entre los dos atributos *saldo*. Se utilizará la operación renombramiento para cambiar el nombre de una referencia a la relación cuenta; así, se puede hacer referencia dos veces a la relación sin ambigüedad alguna.

La relación temporal que se compone de los saldos que no son el máximo puede escribirse ahora como

$$\Pi_{cuenta.saldo} (\sigma_{cuenta.saldo < d.saldo} (cuenta \times \rho_d (cuenta)))$$

Esta expresión proporciona los saldos de la relación *cuenta* para los que aparece un saldo mayor en alguna parte de la relación *cuenta* (cuyo nombre se ha cambiado a *d*). El resultado contiene todos los saldos *salvo* el máximo. Esta relación se muestra en la Figura 3.17.

Paso 2: La consulta para averiguar el máximo saldo de cuenta del banco puede escribirse de la manera siguiente:

$$\Pi_{saldo} (cuenta) - \Pi_{cuenta.saldo} (\sigma_{cuenta.saldo < d.saldo} (cuenta \times \rho_d (cuenta)))$$

En la Figura 3.18 se muestra el resultado de esta consulta.

Considérese la siguiente consulta como un nuevo ejemplo de la operación renombramiento: «Averiguar los nombres de todos los clientes que viven en la misma calle y en la misma ciudad que Gómez». Se puede obtener la calle y la ciudad en la que vive Gómez escribiendo

$$\Pi_{calle\text{-}cliente,\ ciudad\text{-}cliente} (\sigma_{nombre\text{-}cliente = «Gómez»} (cliente))$$

Sin embargo, para hallar a otros clientes que vivan en esa calle y en esa ciudad hay que hacer referencia por segunda vez a la relación *cliente*. En la consulta siguiente se utiliza la operación renombramiento sobre la expresión anterior para darle al resultado el nombre *dirección-Gómez* y para cambiar el nombre de los atributos a *calle* y *ciudad* en lugar de *calle-cliente* y *ciudad-cliente*:

$$\Pi_{cliente.nombre\text{-}cliente} (\sigma_{cliente.calle\text{-}cliente = dirección\text{-}Gómez \wedge cliente.ciudad\text{-}cliente = dirección\text{-}Gómez.\ ciudad} (cliente \times \rho_{dirección\text{-}Gómez\ (calle,\ ciudad)} (\Pi_{calle\text{-}cliente,\ ciudad\text{-}cliente} (\sigma_{nombre\text{-}cliente = «Gómez»} (cliente)))))$$

El resultado de esta consulta, cuando se aplica a la relación *cliente* de la Figura 3.4, se muestra en la Figura 3.19.

La operación renombramiento no es estrictamente necesaria, dado que es posible utilizar una notación posicional para los atributos. Se pueden nombrar los atributos de una relación de manera implícita utilizando una notación posicional, donde \$1, \$2, … hagan refe-

saldo
900

FIGURA 3.18. Saldo máximo de las cuentas del banco.

nombre-cliente
Gómez
Pérez

FIGURA 3.19. Los clientes que viven en la misma calle y en la misma ciudad que Gómez.

rencia al primer atributo, al segundo, etcétera. La notación posicional también se aplica a los resultados de las operaciones del álgebra relacional. La siguiente expresión del álgebra relacional ilustra el uso de la notación posicional con el operador unario σ:

$$\sigma_{\$2=\$3}\,(R \times R)$$

Si una operación binaria necesita distinguir entre las dos relaciones que son sus operandos, se puede utilizar una notación posicional parecida para los nombres de las relaciones. Por ejemplo, $\$R1$ puede hacer referencia al primer operando y $\$R2$, al segundo. Sin embargo, la notación posicional no resulta conveniente para las personas, dado que la posición del atributo es un número en vez de un nombre de atributo fácil de recordar. Por tanto, en este libro no se utiliza la notación posicional.

3.2.2. Definición formal del álgebra relacional

Las operaciones que se vieron en el Apartado 3.2.1 permiten dar una definición completa de las expresiones del álgebra relacional. Las expresiones fundamentales del álgebra relacional se componen de alguna de las siguientes:

- Una relación de la base de datos
- Una relación constante

Una relación constante se escribe listando sus tuplas entre llaves ({}), por ejemplo {(C-101,Centro,500) (C-215, Becerril, 700)}.

Las expresiones generales del álgebra relacional se construyen a partir de subexpresiones menores. Sean E_1 y E_2 expresiones de álgebra relacional. Todas las siguientes son expresiones del álgebra relacional:

- $E_1 \cup E_2$
- $E_1 - E_2$
- $E_1 \times E_2$
- $\sigma_P(E_1)$, donde P es un predicado de atributos de E_1
- $\Pi_S\,(E_1)$, donde S es una lista que se compone de algunos de los atributos de E_1
- $\rho_x\,(E_1)$, donde x es el nuevo nombre del resultado de E_1.

3.2.3. Otras operaciones

Las operaciones fundamentales del álgebra relacional son suficientes para expresar cualquier consulta del álgebra relacional[1]. Sin embargo, si uno se limita únicamente a las operaciones fundamentales, algunas consultas habituales resultan de expresión intrincada. Por tanto, se definen otras operaciones que no añaden potencia al álgebra, pero que simplifican las consultas habituales. Para cada operación nueva se facilita una expresión equivalente utilizando sólo las operaciones fundamentales.

3.2.3.1. La operación intersección de conjuntos

La primera operación adicional del álgebra relacional que se definirá es la **intersección de conjuntos** ($\cap$). Supóngase que se desea averiguar todos los clientes que tienen un préstamo concedido y una cuenta abierta. Utilizando la intersección de conjuntos se puede escribir

$$\Pi_{nombre\text{-}cliente}\,(prestatario) \cap \Pi_{nombre\text{-}cliente}\,(impositor)$$

La relación resultante de esta consulta aparece en la Figura 3.20.

Obsérvese que se puede volver a escribir cualquier expresión del álgebra relacional utilizando la intersección de conjuntos sustituyendo la operación intersección por un par de operaciones de diferencia de conjuntos, de la manera siguiente:

$$r \cap s = r - (r - s)$$

Por tanto, la intersección de conjuntos no es una operación fundamental y no añade potencia al álgebra relacional. Sencillamente, es más conveniente escribir $r \cap s$ que $r - (r - s)$.

3.2.3.2. La operación reunión natural

Suele resultar deseable simplificar ciertas consultas que exigen un producto cartesiano. Generalmente, las consultas que implican un producto cartesiano incluyen un operador selección sobre el resultado del producto cartesiano. Considérese la consulta «Hallar los nombres de todos los clientes que tienen concedido un préstamo en el banco y averiguar el importe del mismo». En primer lugar se calcula el producto cartesiano de las relaciones *prestatario* y *préstamo*. Luego, se seleccionan las tuplas que sólo atañen al mismo *número-préstamo*, seguidas por la proyección de *nombre-cliente*, *número-préstamo* e *importe* resultantes:

$$\Pi_{nombre\text{-}cliente,\ préstamo.número\text{-}préstamo,\ importe}\\(\sigma_{prestatario.número\text{-}préstamo\ =\ préstamo.número\text{-}préstamo}\\(prestatario \times préstamo))$$

[1] En el Apartado 3.3 se introducen las operaciones que extienden la potencia del álgebra relacional al tratamiento de los valores nulos y los valores de agregación.

nombre-cliente
Gómez
Pérez
Santos

FIGURA 3.20. Clientes con una cuenta abierta y un préstamo en el banco

La *reunión natural* es una operación binaria que permite combinar ciertas selecciones y un producto cartesiano en una sola operación. Se denota por el símbolo de la «reunión» $\bowtie$. La operación reunión natural forma un producto cartesiano de sus dos argumentos, realiza una selección forzando la igualdad de los atributos que aparecen en ambos esquemas de relación y, finalmente, elimina los atributos duplicados.

Aunque la definición de la reunión natural es compleja, la operación es sencilla de aplicar. Como ilustración, considérese nuevamente el ejemplo «Averiguar los nombres de todos los clientes que tienen concedido un préstamo en el banco y averiguar su importe». Esta consulta puede expresarse utilizando la reunión natural de la manera siguiente:

$$\Pi_{nombre\text{-}cliente,\ número\text{-}préstamo,\ importe}\ (prestatario \bowtie préstamo)$$

Dado que los esquemas de *prestatario* y de *préstamo* (es decir, *Esquema-prestatario* y *Esquema-préstamo*) tienen en común el atributo *número-préstamo*, la operación reunión natural sólo considera los pares de tuplas que tienen el mismo valor de *número-préstamo*. Esta operación combina cada uno de estos pares en una sola tupla en la unión de los dos esquemas (es decir, *nombre-cliente*, *nombre-sucursal*, *número-préstamo*, *importe*). Después de realizar la proyección, se obtiene la relación mostrada en la Figura 3.21.

Considérense dos esquemas de relación R y S que son, por supuesto, listas de nombres de atributos. Si se consideran los esquemas como *conjuntos*, en vez de como listas, se pueden denotar los nombres de los atributos que aparecen tanto en R como en S mediante $R \cap S$, y los nombres de los atributos que aparecen en R, en S o en ambos mediante $R \cup S$. De manera parecida, los nombres de los atributos que aparecen en R pero no en S se denotan por $R - S$, mientras que $S - R$ denota los nombres de los atributos que aparecen en S pero no en R. Obsérvese que las operaciones unión, intersección y diferencia aquí operan sobre conjuntos de atributos, y no sobre relaciones.

nombre-cliente	*número-préstamo*	*importe*
Fernández	P-16	1.300
Gómez	P-23	2.000
Gómez	P-11	900
López	P-15	1.500
Pérez	P-93	500
Santos	P-17	1.000
Sotoca	P-14	1.500
Valdivieso	P-17	1.000

FIGURA 3.21. Resultado de $\Pi_{nombre\text{-}cliente,\ número\text{-}préstamo,\ importe}$ (*prestatario* $\bowtie$ *préstamo*).

Ahora se está preparado para una definición formal de la reunión natural. Considérense dos relaciones $r(R)$ y $s(S)$. La **reunión natural** de r y de s, denotada por $r \bowtie s$ es una relación del esquema $R \cup S$ definida formalmente de la manera siguiente:

$$r \bowtie s = \Pi_{R \cup S}\,(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n}\ r \times s)$$

donde $R \cap S = \{A_1, A_2, \ldots, A_n\}$.

Como la reunión natural es fundamental para gran parte de la teoría y de la práctica de las bases de datos relacionales, se ofrecen varios ejemplos de su uso.

- Hallar los nombres de todas las sucursales con clientes que tienen una cuenta abierta en el banco y que viven en Peguerinos.

  $$\Pi_{nombre\text{-}sucursal}\ (\sigma_{ciudad\text{-}cliente = «Peguerinos»}\ (cliente \bowtie cuenta \bowtie impositor))$$

  La relación resultante de esta consulta aparece en la Figura 3.22.

  Obsérvese que se escribió *cliente* $\bowtie$ *cuenta* $\bowtie$ *impositor* sin añadir paréntesis para especificar el orden en que se deben ejecutar las operaciones reunión natural de las tres relaciones. En el caso anterior hay dos posibilidades:

  — (*cliente* $\bowtie$ *cuenta*) $\bowtie$ *impositor*
  — *cliente* $\bowtie$ (*cuenta* $\bowtie$ *impositor*)

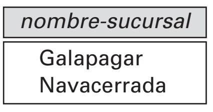
  No se especificó la expresión deseada porque las dos son equivalentes. Es decir, la reunión natural es **asociativa**.

- Hallar todos los clientes que tienen una cuenta abierta *y* un préstamo concedido en el banco.

  $$\Pi_{nombre\text{-}cliente}\ (prestatario \bowtie impositor)$$

  Obsérvese que en el Apartado 3.2.3.1 se escribió una expresión para esta consulta utilizando la intersección de conjuntos. Aquí se repite esa expresión.

$$\Pi_{nombre\text{-}cliente}\ (prestatario) \cap \Pi_{nombre\text{-}cliente}\ (impositor)$$



nombre-sucursal
Galapagar
Navacerrada

FIGURA 3.22. Resultado de $\Pi_{nombre\text{-}sucursal}$ ($\sigma_{ciudad\text{-}cliente = «Peguerinos»}$ (*cliente* $\bowtie$ *cuenta* $\bowtie$ *impositor*)).

La relación resultante de esta consulta se mostró anteriormente en la Figura 3.20. Este ejemplo ilustra una realidad común del álgebra relacional: se pueden escribir varias expresiones del álgebra relacional equivalentes que sean bastante diferentes entre sí.

- Sean $r(R)$ y $s(S)$ relaciones sin atributos en común; es decir, $R \cap S = \emptyset$. ($\emptyset$ denota el conjunto vacío.) Por tanto, $r \bowtie s = r \times s$.

La operación *reunión zeta* es una extensión de la operación reunión natural que permite combinar una selección y un producto cartesiano en una sola operación. Considérense las relaciones $r(R)$ y $s(S)$, y sea θ un predicado de los atributos del esquema $R \cup S$. La operación **reunión zeta** $r \bowtie_{\theta} s$ se define de la manera siguiente:

$$r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$$

3.2.3.3. La operación división

La operación **división**, denotada por ÷, resulta adecuada para las consultas que incluyen la expresión «para todos». Supóngase que se desea hallar a todos los clientes que tengan abierta una cuenta en *todas* las sucursales ubicadas en Arganzuela. Se pueden obtener todas las sucursales de Arganzuela mediante la expresión

$$r_1 = \Pi_{nombre\text{-}sucursal}(\sigma_{ciudad\text{-}sucursal = \text{«Arganzuela»}}(sucursal))$$

La relación resultante de esta expresión aparece en la Figura 3.23.

Se pueden encontrar todos los pares (*nombre-cliente*, *nombre-sucursal*) para los que el cliente tiene una cuenta en una sucursal escribiendo

$$r_2 = \Pi_{nombre\text{-}cliente,\ nombre\text{-}sucursal}(impositor \bowtie cuenta)$$

La Figura 3.24 muestra la relación resultante de esta expresión.

Ahora hay que hallar los clientes que aparecen en r_2 con los nombres de *todas* las sucursales de r_1. La operación que proporciona exactamente esos clientes es la operación división. La consulta se formula escribiendo

$$\Pi_{nombre\text{-}cliente,\ nombre\text{-}sucursal}(impositor \bowtie cuenta) \div \Pi_{nombre\text{-}sucursal}(\sigma_{ciudad\text{-}sucursal = \text{«Arganzuela»}}(sucursal))$$

El resultado de esta expresión es una relación que tiene el esquema (*nombre-cliente*) y que contiene la tupla (González).

nombre-sucursal
Centro
Galapagar

FIGURA 3.22. Resultado de $\Pi_{nombre\text{-}sucursal}(\sigma_{ciudad\text{-}sucursal = \text{«Arganzuela»}}(sucursal))$.

nombre-cliente	*nombre-sucursal*
Abril	Collado Mediano
Gómez	Becerril
González	Centro
González	Galapagar
López	Navacerrada
Rupérez	Moralzarzal
Santos	Galapagar
Valdivieso	Navacerrada

FIGURA 3.24. Resultado de $\Pi_{nombre\text{-}cliente,\ nombre\text{-}sucursal}(impositor \bowtie cuenta)$).

Formalmente, sean $r(R)$ y $s(S)$ relaciones y $S \subseteq R$; es decir, todos los atributos del esquema S están también en el esquema R. La relación $r \div s$ es una relación del esquema $R - S$ (es decir, del esquema que contiene todos los atributos del esquema R que no están en el esquema S). Una tupla t está en $r \div s$ si y sólo si se cumplen estas dos condiciones:

1. t está en $\Pi_{R-S}(r)$
2. Para cada tupla t_S de s hay una tupla t_r de r que cumple las dos condiciones siguientes:
   a. $t_r[S] = t_s[S]$
   b. $t_r[R - S] = t$

Puede resultar sorprendente descubrir que, dados una operación división y los esquemas de las relaciones, se puede, de hecho, definir la operación división en términos de las operaciones fundamentales. Sean $r(R)$ y $s(S)$ dadas, con $S \subseteq R$:

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

Para comprobar que esta expresión es verdadera, obsérvese que $\Pi_{R-S}(r)$ da todas las tuplas t que cumplen la primera condición de la definición de la división. La expresión del lado derecho del operador diferencia de conjuntos,

$$\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)),$$

sirve para borrar esas tuplas que no cumplen la segunda condición de la definición de la división. Esto se logra de la manera siguiente. Considérese $\Pi_{R-S}(r) \times s$. Esta relación está en el esquema R y empareja cada tupla de $\Pi_{R-S}(r)$ con cada tupla de s. La expresión $\Pi_{R-S,S}(r)$ sólo reordena los atributos de r.

Por tanto, $(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)$ genera los pares de tuplas de $\Pi_{R-S}(r)$ y de s que no aparecen en r. Si una tupla t_j está en

$$\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)),$$

hay alguna tupla t_s de s que no se combina con la tupla t_j para formar una tupla de r. Por tanto, t_j guarda un valor de los atributos $R - S$ que no aparece en $r \div s$. Estos valores son los que se eliminan de $\Pi_{R-S}(r)$.

3.2.3.4. La operación asignación

En ocasiones resulta conveniente escribir una expresión del álgebra relacional por partes utilizando la asignación a una variable de relación temporal. La operación **asignación**, denotada por ←, actúa de manera parecida a la asignación de los lenguajes de programación. Para ilustrar esta operación, considérese la definición de la división dada en el Apartado 3.2.3.3. Se puede escribir $r \div s$ como

$$temp1 \leftarrow \Pi_{R-S}(r)$$
$$temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$$
$$resultado = temp1 - temp2$$

La evaluación de una asignación no hace que se muestre ninguna relación al usuario. Por el contrario, el resultado de la expresión a la derecha de ← se asigna a la variable relación a la izquierda de ←. Esta variable relación puede utilizarse en expresiones posteriores.

Con la operación asignación se pueden escribir las consultas como programas secuenciales consistentes en una serie de asignaciones seguida de una expresión cuyo valor se muestra como resultado de la consulta. En las consultas del álgebra relacional la asignación siempre debe hacerse a una variable de relación intermedia. Las asignaciones a relaciones permanentes constituyen una modificación de la base de datos. Este asunto se discutirá en el Apartado 3.4. Obsérvese que la operación asignación no añade potencia alguna al álgebra. Resulta, sin embargo, una manera conveniente de expresar las consultas complejas.

3.3. OPERACIONES DEL ÁLGEBRA RELACIONAL EXTENDIDA

Las operaciones básicas del álgebra relacional se han ampliado de varias maneras. Una ampliación sencilla es permitir operaciones aritméticas como parte de la proyección. Una ampliación importante es permitir *operaciones de agregación*, como el cálculo de la suma de los elementos de un conjunto, o su media. Otra ampliación importante es la operación *reunión externa*, que permite a las expresiones del álgebra relacional trabajar con los valores nulos que modelan la información que falta.

3.3.1. Proyección generalizada

La operación **proyección generalizada** amplía la operación proyección permitiendo que se utilicen funciones aritméticas en la lista de proyección. La operación proyección generalizada tiene la forma

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

donde E es cualquier expresión del álgebra relacional y $F_1, F_2, \ldots, F_n$ son expresiones aritméticas que incluyen constantes y atributos en el esquema de E. Como caso especial la expresión aritmética puede ser simplemente un atributo o una constante.

Por ejemplo, supóngase que se dispone de una relación *información-crédito*, como se muestra en la Figura 3.25, que da el límite de crédito y el importe dispuesto hasta el momento presente (el *saldo-crédito* de la cuenta). Si se desea averiguar el importe disponible por cada persona, se puede escribir la expresión siguiente:

$$\Pi_{nombre\text{-}cliente,\ límite\ -\ saldo\text{-}crédito}(información\text{-}crédito)$$

El atributo resultante de la expresión *límite – saldo-crédito* no tiene un nombre. Se puede aplicar la operación renombramiento al resultado de la proyección generalizada para darle un nombre. Como conveniencia notacional, el renombramiento de atributos se puede combinar con la proyección generalizada como se ilustra a continuación:

$$\Pi_{nombre\text{-}cliente,\ (límite\ -\ saldo\text{-}crédito)\ \mathbf{as}\ crédito\text{-}disponible}(información\text{-}crédito)$$

Al segundo atributo de esta proyección generalizada se le ha dado el nombre *crédito-disponible*. En la Figura 3.26 se muestra el resultado de aplicar esta expresión a la relación de la Figura 3.25.

3.3.2. Funciones de agregación

Las **funciones de agregación** son funciones que toman una colección de valores y devuelven como resultado un único valor. Por ejemplo, la función de agregación

nombre-cliente	*límite*	*saldo-crédito*
Gómez	2.000	400
López	1.500	1.500
Pérez	2.000	1.750
Santos	6.000	700

FIGURA 3.25. La relación *información-crédito*.

nombre-cliente	*crédito-disponible*
Gómez	1.600
López	0
Pérez	250
Santos	5.300

FIGURA 3.26. Resultado de $\Pi_{nombre\text{-}cliente,\ (límite\ -\ saldo\text{-}crédito)\ \mathbf{as}\ crédito\text{-}disponible}(información\text{-}crédito)$.

sum toma un conjunto de valores y devuelve la suma de los mismos. Por tanto, la función **sum** aplicada a la colección

$$\{1, 1, 3, 4, 4, 11\}$$

devuelve el valor 24. La función de agregación **avg** devuelve la media de los valores. Cuando se aplica al conjunto anterior devuelve el valor 4. La función de agregación **count** devuelve el número de elementos del conjunto, y devolvería 6 en el caso anterior. Otras funciones de agregación habituales son **min** y **max**, que devuelven el valor mínimo y el máximo de la colección; en el ejemplo anterior devuelven 1 y 11, respectivamente.

Las colecciones en las que operan las funciones de agregación pueden tener valores repetidos; el orden en el que aparezcan los valores no tiene importancia. Estas colecciones se denominan **multiconjuntos**. Los conjuntos son un caso especial de los multiconjuntos, en los que sólo hay una copia de cada elemento.

Para ilustrar el concepto de agregación se utilizará la relación *trabajo-por-horas* descrita en la Figura 3.27, que muestra los empleados a tiempo parcial. Supóngase que se desea averiguar la suma total de los sueldos de los empleados del banco a tiempo parcial. La expresión del álgebra relacional para esta consulta es:

$$\mathcal{G}_{\mathbf{sum}(sueldo)}\,(trabajo\text{-}por\text{-}horas)$$

El símbolo $\mathcal{G}$ es la letra G en el tipo de letra caligráfico; se lee «G caligráfica». La operación del álgebra relacional $\mathcal{G}$ significa que se debe aplicar agregación, y el subíndice indica la operación de agregación a aplicar. El resultado de la expresión anterior es una relación con un único atributo, que contiene una sola fila con un valor correspondiente a la suma de los sueldos de todos los trabajadores que trabajan en el banco a tiempo parcial.

Hay casos en los que se deben borrar los valores repetidos antes de calcular una función de agregación. Si se desean borrar los valores repetidos hay que utilizar los mismos nombres de funciones que antes, con la cadena de texto «**distinct**» precedida de un guión añadida al final del nombre de la función (por ejemplo, **count-distinct**). Un ejemplo se da en la consulta «Averiguar el número de sucursales que aparecen en la relación *trabajo-por-horas*». En este caso, el nombre de cada sucursal sólo se cuenta una vez, independientemente del número de empleados que trabajen en la misma. Esta consulta se escribe de la manera siguiente:

$$\mathcal{G}_{\mathbf{count\text{-}distinct}(nombre\text{-}sucursal)}\,(trabajo\text{-}por\text{-}horas)$$

Para la relación mostrada en la Figura 3.27 el resultado de esta consulta es el valor 3.

Supóngase que se desea hallar la suma total de sueldos de todos los empleados a tiempo parcial en cada sucursal del banco por separado, en lugar de hallar la suma de sueldos de todo el banco. Para ello hay que dividir la relación *trabajo-por-horas* en **grupos** basados en la sucursal y aplicar la función de agregación a cada grupo.

La expresión siguiente obtiene el resultado deseado utilizando el operador de agregación $\mathcal{G}$:

$$_{nombre\text{-}sucursal}\mathcal{G}_{\mathbf{sum}(sueldo)}\,(trabajo\text{-}por\text{-}horas)$$

El atributo *nombre-sucursal* subíndice a la izquierda de *G* indica que la relación de entrada *trabajo-por-horas* debe dividirse en grupos de acuerdo con el valor de *nombre-sucursal*. Los grupos resultantes se muestran en la Figura 3.28. La expresión **sum**(*sueldo*) en el subíndice derecho de $\mathcal{G}$ indica que, para cada grupo de tuplas (es decir, para cada sucursal) hay que aplicar la función de agregación **sum** al conjunto de valores del atributo *sueldo*. La relación resultante consiste en las tuplas con el nombre de la sucursal y la suma de los sueldos de la sucursal, como se muestra en la Figura 3.29.

La forma general de la **operación de agregación** $\mathcal{G}$ es la siguiente:

$$_{G_1, G_2, \ldots, G_n}\mathcal{G}_{F_1(A_1), F_2(A_2), \ldots, F_m(A_m)}\,(E)$$

donde *E* es cualquier expresión del álgebra relacional; $G_1, G_2, \ldots, G_n$ constituye una lista de atributos que indican cómo se realiza la agrupación, cada F_i es una función de agregación y cada A_i es el nombre de un atributo. El significado de la operación se define de la manera siguiente. Las tuplas en el resultado de la expresión *E* se dividen en grupos tales que

nombre-empleado	*nombre-sucursal*	*sueldo*
González	Centro	1.500
Díaz	Centro	1.300
Jiménez	Centro	2.500
Catalán	Leganés	1.600
Cana	Leganés	1.500
Cascallar	Navacerrada	5.300
Fernández	Navacerrada	1.500
Ribera	Navacerrada	1.300

FIGURA 3.27. La relación *trabajo-por-horas*.

nombre-empleado	*nombre-sucursal*	*sueldo*
González	Centro	1.500
Díaz	Centro	1.300
Jiménez	Centro	2.500
Catalán	Leganés	1.600
Cana	Leganés	1.500
Cascallar	Navacerrada	5.300
Fernández	Navacerrada	1.500
Ribera	Navacerrada	1.300

FIGURA 3.28. La relación *trabajo-por-horas* después de la agrupación.

nombre-sucursal	*suma de sueldos*
Centro	5.300
Leganés	3.100
Navacerrada	8.100

FIGURA 3.29. Resultado de $_{\textit{nombre-sucursal}}\mathcal{G}_{\textbf{sum}(\text{sueldo})}$ (*trabajo-por-horas*).

1. Todas las tuplas del grupo tienen los mismos valores para $G_1, G_2, \ldots, G_n$.
2. Las tuplas de grupos diferentes tienen valores diferentes para $G_1, G_2, \ldots, G_n$.

Por tanto, los grupos pueden identificarse por el valor de los atributos $G_1, G_2, \ldots, G_n$. Para cada grupo $(g_1, g_2, \ldots, g_n)$ el resultado tiene una tupla $(g_1, g_2, \ldots, g_n, a_1, a_2, \ldots, a_m)$ donde, para cada i, a_i es el resultado de aplicar la función de agregación F_i al multiconjunto de valores del atributo A_i en el grupo.

Como caso especial de la operación de agregación, la lista de atributos $G_1, G_2, \ldots, G_n$ puede estar vacía, en cuyo caso sólo hay un grupo que contiene todas las tuplas de la relación. Esto corresponde a la agregación sin agrupación.

Volviendo al ejemplo anterior, si se deseara averiguar el sueldo máximo de los empleados a tiempo parcial de cada oficina, además de la suma de los sueldos, habría que escribir la expresión

$$_{\textit{nombre-sucursal}}\mathcal{G}_{\textbf{sum}(\textit{sueldo}),\ \textbf{max}(\textit{sueldo})}(\textit{trabajo-por-horas})$$

Como en la proyección generalizada, el resultado de una operación de agregación no tiene nombre. Se puede aplicar la operación renombramiento al resultado para darle un nombre. Como conveniencia notacional, los atributos de una operación de agregación se pueden renombrar como se indica a continuación:

$$_{\textit{nombre-sucursal}}\mathcal{G}_{\textbf{sum}(\textit{sueldo})\ \textbf{as}\ \textit{suma-sueldo},\textbf{max}(\textit{sueldo})\ \textbf{as}\ \textit{sueldo-máximo}}(\textit{trabajo-por-horas})$$

El resultado de la expresión se muestra en la Figura 3.30.

3.3.3. Reunión externa

La operación **reunión externa** es una ampliación de la operación reunión para trabajar con la información que falta. Supóngase que se dispone de relaciones con los siguientes esquemas, que contienen datos de empleados a tiempo completo:

empleado (*nombre-empleado, calle, ciudad*)
trabajo-a-tiempo-completo (*nombre-empleado, nombre-sucursal, sueldo*)

Considérense las relaciones *empleado* y *trabajo-a-tiempo-completo* mostradas en la Figura 3.31. Supóngase que se desea generar una única relación con toda la información (calle, ciudad, nombre de la sucursal y sueldo) de los empleados a tiempo completo. Un posible enfoque sería utilizar la operación reunión natural de la manera siguiente:

$$\textit{empleado} \bowtie \textit{trabajo-a-tiempo-completo}$$

El resultado de esta expresión se muestra en la Figura 3.32. Obsérvese que se ha perdido la información sobre la calle y la ciudad de residencia de Gómez, dado que la tupla que describe a Gómez no está presente en la relación *trabajo-a-tiempo-completo*; de manera parecida, se ha perdido la información sobre el nombre de la sucursal y sobre el sueldo de Barea, dado que la tupla que describe a Barea no está presente en la relación *empleado*.

Se puede utilizar la operación reunión externa para evitar esta pérdida de información. En realidad, esta operación tiene tres formas diferentes: *reunión externa por la izquierda*, denotada por ⟕; *reunión externa por la derecha*, denotada por ⟖ y *reunión externa completa*, denotada por ⟗. Las tres formas de la reunión externa calculan la reunión y añaden tuplas adicionales al resultado de la misma. El resultado de las expresiones *empleado* ⟕ *trabajo-a-tiempo-completo*, *empleado* ⟖ *trabajo-a-tiempo-completo* y *empleado* ⟗ *trabajo-a-tiempo-completo* se muestra en las Figuras 3.33, 3.34 y 3.35, respectivamente.

La **reunión externa por la izquierda** (⟕) toma todas las tuplas de la relación de la izquierda que no coincidan con ninguna tupla de la relación de la derecha, las rellena con valores nulos en todos los demás atributos de la relación de la derecha y las añade al resul-

nombre-sucursal	*suma-sueldo*	*sueldo-máximo*
Centro	5.300	2.500
Leganés	3.100	1.600
Navacerrada	8.100	5.300

FIGURA 3.30. Resultado de $_{\textit{nombre-sucursal}}\mathcal{G}_{\textbf{sum}(\text{sueldo})\ \textbf{as}\ \text{suma-sueldo}\ \textbf{max}(\text{sueldo})\ \textbf{as}\ \text{sueldo-máximo}}$ (*trabajo-por-horas*).

nombre-empleado	*calle*	*ciudad*
Segura	Tebeo	La Loma
Domínguez	Viaducto	Villaconejos
Gómez	Bailén	Alcorcón
Valdivieso	Fuencarral	Móstoles

nombre-empleado	*nombre-sucursal*	*sueldo*
Segura	Majadahonda	1.500
Domínguez	Majadahonda	1.300
Barea	Fuenlabrada	5.300
Valdivieso	Fuenlabrada	1.500

FIGURA 3.31. Las relaciones *empleado* y *trabajo-a-tiempo-completo.*

nombre-empleado	*calle*	*ciudad*	*nombre-sucursal*	*sueldo*
Segura	Tebeo	La Loma	Majadahonda	1.500
Domínguez	Viaducto	Villaconejos	Majadahonda	1.300
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	1.500

FIGURA 3.32. La relación *empleado* ⋈ *trabajo-a-tiempo-completo.*

tado de la reunión natural. En la Figura 3.33 la tupla (Gómez, Bailén, Alcorcón, *nulo*, *nulo*) es una tupla de este tipo. Toda la información de la relación de la izquierda se halla presente en el resultado de la reunión externa por la izquierda.

La **reunión externa por la derecha** (⟖) es simétrica de la reunión externa por la izquierda. Las tuplas de la relación de la derecha que no coincidan con ninguna tupla de la relación de la izquierda se rellenan con valores nulos y se añaden al resultado de la reunión natural. En la Figura 3.34 la tupla (Barea, *nulo*, *nulo*, Fuenlabrada, 5.300) es una tupla de este tipo. Por tanto, toda la información de la relación de la derecha se halla presente en el resultado de la reunión externa por la derecha.

nombre-empleado	*calle*	*ciudad*	*nombre-sucursal*	*sueldo*
Segura	Tebeo	La Loma	Majadahonda	1.500
Domínguez	Viaducto	Villaconejos	Majadahonda	1.300
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	1.500
Gómez	Bailén	Alcorcón	*nulo*	*nulo*

FIGURA 3.33. Resultado de *empleado* ⟕ *trabajo-a-tiempo-completo.*

La **reunión externa completa** (⟗) realiza estas dos operaciones, rellenando las tuplas de la relación de la izquierda que no coincidan con ninguna tupla de la relación de la derecha y las tuplas de la relación de la derecha que no coincidan con ninguna tupla de la relación de la izquierda, y añadiéndolas al resultado de la reunión. En la Figura 3.35 se muestra el resultado de una reunión externa completa.

Puesto que las operaciones de reunión pueden generar resultados que contengan nulos, es necesario especificar cómo deben manejar estos valores las operaciones del álgebra relacional. El Apartado 3.3.4 aborda este aspecto.

Es interesante observar que las operaciones de reunión externa pueden expresar mediante las operaciones básicas del álgebra relacional. Por ejemplo, la operación de reunión externa por la izquierda r ⟕ s se puede expresar como:

$$(r \bowtie s) \cup (r - \Pi_R(r \bowtie s)) \times \{(nulo, \ldots, nulo)\}$$

donde la relación constante $\{(nulo, \ldots, nulo)\}$ se encuentra en el esquema $S - R$.

nombre-empleado	*calle*	*ciudad*	*nombre-sucursal*	*sueldo*
Segura	Tebeo	La Loma	Majadahonda	1.500
Domínguez	Viaducto	Villaconejos	Majadahonda	1.300
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	1.500
Barea	*nulo*	*nulo*	Fuenlabrada	5.300

FIGURA 3.34. Resultado de *empleado* ⟖ *trabajo-a-tiempo-completo.*

3.3.4. Valores nulos**

En este apartado se define la forma en que las diferentes operaciones del álgebra relacional tratan los valores nulos y las complicaciones que surgen cuando los valores nulos participan en las operaciones aritméticas o en

nombre-empleado	*calle*	*ciudad*	*nombre-sucursal*	*sueldo*
Segura	Tebeo	La Loma	Majadahonda	1.500
Domínguez	Viaducto	Villaconejos	Majadahonda	1.300
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	1.500
Gómez	Bailén	Alcorcón	*nulo*	*nulo*
Barea	*nulo*	*nulo*	Fuenlabrada	5.300

FIGURA 3.35. Resultado de *empleado* ⟗ *trabajo-a-tiempo-completo.*

las comparaciones. Como se verá, a menudo hay varias formas de tratar los valores nulos y, como resultado, las siguientes definiciones pueden ser a veces arbitrarias. Las operaciones y las comparaciones con valores nulos se deberían evitar siempre que sea posible.

Dado que el valor especial *nulo* indica «valor desconocido o no existente», cualquier operación aritmética (como +, –, ∗ y /) que incluya valores nulos debe devolver un valor nulo.

De manera similar, cualquier comparación (como <, <=, >, >= y ≠) que incluya un valor nulo se evalúa al valor especial **desconocido**; no se puede decir si el resultado de la comparación es cierto o falso, así que se dice que el resultado es el nuevo valor lógico *desconocido*.

Las comparaciones que incluyan nulos pueden aparecer dentro de expresiones booleanas que incluyan las operaciones y (conjunción), o (disyunción) y no (negación). Se debe definir la forma en que estas operaciones tratan el valor lógico *desconocido*.

- **y**: (*cierto* **y** *desconocido*) = *desconocido*; (*falso* **y** *desconocido*) = *falso*; (*desconocido* **y** *desconocido*) = *desconocido*.
- **o**: (*cierto* **o** *desconocido*) = *cierto*; (*falso* **o** *desconocido*) = *desconocido*; (*desconocido* **o** *desconocido*) = *desconocido*.
- **no**: (**no** *desconocido*) = *desconocido*.

Ahora es posible describir la forma en que las diferentes operaciones del álgebra relacional tratan los valores nulos. Nuestras definiciones siguen las usadas en el lenguaje SQL.

- **select**: la operación selección evalúa el predicado P en $\sigma_P(E)$ sobre cada tupla de E. Si el predicado devuelve el valor *cierto*, se añade t al resultado. En caso contrario, si el predicado devuelve *desconocido* o *falso*, t no se añade al resultado.
- **reunión**: las reuniones se pueden expresar como un producto cartesiano seguido de una selección. Por tanto, la definición de la forma en que la selección trata los nulos también define la forma en que la operación reunión trata los nulos.

  En una reunión natural $r \bowtie s$ se puede observar de la definición anterior que si dos tuplas, $t_r \in r$ y $t_s \in s$, tienen un valor nulo en un atributo común, entonces las tuplas no casan.
- **proyección**: la operación proyección trata los nulos como cualquier otro valor al eliminar duplicados. Así, si dos tuplas del resultado de la proyección son exactamente iguales, y ambos tienen nulos en los mismos campos, se tratan como duplicados.

  La decisión es un tanto arbitraria porque sin saber cuál es el valor real no se sabe si los dos valores nulos son duplicados o no.
- **unión, intersección, diferencia**: estas operaciones tratan los valores nulos al igual que la operación proyección; tratan las tuplas que tienen los mismos valores en todos los campos como duplicados incluso si algunos de los campos tienen valores nulos en ambas tuplas.

  El comportamiento es un tanto arbitrario, especialmente en el caso de la intersección y la diferencia, dado que no se sabe si los valores reales (si existen) representados por los nulos son los mismos.
- **proyección generalizada**: se describió la manera en que se tratan los nulos en las expresiones al principio del Apartado 3.3.4. Las tuplas duplicadas que contienen valores nulos se tratan como en la operación proyección.

  Cuando hay nulos en los atributos agregados, la operación borra los valores nulos del resultado antes de aplicar la agregación. Si el multiconjunto resultante está vacío, el resultado agregado es nulo.

  Obsérvese que el tratamiento de los valores nulos aquí es diferente que en las expresiones aritméticas ordinarias; se podría haber definido el resultado de una operación de agregación como nulo si incluso sólo uno de los valores agregados es nulo. Sin embargo, esto significaría que un único valor desconocido en un gran grupo podría hacer que el resultado agregado sobre el grupo fuese nulo, y se perdería una gran cantidad de información útil.
- **reunión externa**: las operaciones de reunión externa se comportan como las operaciones reunión, excepto sobre las tuplas que no aparecen en el resultado. Estas tuplas se pueden añadir al resultado (dependiendo de si la operación es ⟕, ⟖ o ⟗) añadiendo nulos.

3.4. MODIFICACIÓN DE LA BASE DE DATOS

Hasta ahora hemos centrado la atención en la extracción de información de la base de datos. En este apartado se abordará la manera de insertar, borrar o modificar información de la base de datos.

Las modificaciones de la base de datos se expresan utilizando la operación asignación. Las asignaciones a las relaciones reales de la base de datos se realizan utilizando la misma notación que se describió para la asignación en el Apartado 3.2.3.

3.4.1. Borrado

Las solicitudes de borrado se expresan básicamente igual que las consultas. Sin embargo, en lugar de mostrar las

tuplas al usuario, se eliminan de la base de datos las tuplas seleccionadas. Sólo se pueden borrar tuplas enteras; no se pueden borrar valores de atributos concretos. En el álgebra relacional los borrados se expresan mediante

$$r \leftarrow r - E$$

donde *r* es una relación y *E* es una consulta del álgebra relacional.

He aquí varios ejemplos de solicitudes de borrado del álgebra relacional:

- Borrar todas las cuentas de Gómez.

$$impositor \leftarrow impositor - \sigma_{nombre\text{-}cliente = «Gómez»}(impositor)$$

- Borrar todos los préstamos con importes entre 0 y 50.

$$préstamo \leftarrow préstamo - \sigma_{importe \geq 0 \text{ and } importe \leq 50}(préstamo)$$

- Borrar todas las cuentas de las sucursales sitas en Getafe.

$$r_1 \leftarrow \sigma_{ciudad\text{-}sucursal = «Getafe»}(cuenta \bowtie sucursal)$$
$$r_2 \leftarrow \Pi_{nombre\text{-}sucursal,\ número\text{-}cuenta,\ saldo}(r_1)$$
$$cuenta \leftarrow cuenta - r_2$$

Obsérvese que en el último ejemplo se simplificó la expresión utilizando la asignación a las relaciones temporales (r_1 y r_2).

3.4.2. Inserción

Para insertar datos en una relación hay que especificar la tupla que se va a insertar o escribir una consulta cuyo resultado sea un conjunto de tuplas que vayan a insertarse. Evidentemente, el valor de los atributos de las tuplas insertadas deben ser miembros del dominio de cada atributo. De manera parecida, las tuplas insertadas deben ser de la aridad correcta. En el álgebra relacional las inserciones se expresan mediante

$$r \leftarrow r \cup E$$

donde *r* es una relación y *E* es una expresión del álgebra relacional. La inserción de una sola tupla se expresa haciendo que *E* sea una relación constante que contiene una tupla.

Supóngase que se desea insertar el hecho de que Gómez tiene 1.200 € en la cuenta C-973 en la sucursal de Navacerrada. Hay que escribir

$$cuenta \leftarrow cuenta \cup \{(\text{C-973}, «\text{Navacerrada}», 1200)\}$$
$$impositor \leftarrow impositor \cup \{(«\text{Gómez}», \text{C-973})\}$$

De forma más general, puede que se desee insertar tuplas de acuerdo con el resultado de una consulta. Supóngase que se desea ofrecer una nueva cuenta de ahorro con 200 € como regalo a todos los clientes con préstamos concedidos en la sucursal de Navacerrada. Sea el número de préstamo el que se utilice como número de cuenta de esta cuenta de ahorro. Hay que escribir

$$r_1 \leftarrow (\sigma_{nombre\text{-}sucursal = «Navacerrada»}(prestatario \bowtie préstamo))$$
$$r_2 \leftarrow \Pi_{nombre\text{-}sucursal,\ número\text{-}préstamo}(r_1)$$
$$cuenta \leftarrow cuenta \cup (r_2 \times \{(200)\})$$
$$impositor \leftarrow impositor \cup \Pi_{nombre\text{-}cliente,\ número\text{-}préstamo}(r_1)$$

En lugar de especificar las tuplas como se hizo anteriormente, se especifica un conjunto de tuplas que se insertan en las relaciones *cuenta* e *impositor*. Cada tupla de la relación *cuenta* tiene el *nombre-sucursal* (Navacerrada), un *número-cuenta* (que es igual que el número de préstamo) y el saldo inicial de la nueva cuenta (200 €). Cada tupla de la relación *impositor* tiene como *nombre-cliente* el nombre del prestatario al que se le da la nueva cuenta y el mismo número de cuenta que la correspondiente tupla de *cuenta*.

3.4.3. Actualización

Puede que, en algunas situaciones, se desee modificar un valor de una tupla sin modificar *todos* los valores de la tupla. Se puede utilizar el operador proyección generalizada para realizar esta tarea:

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(r)$$

donde cada F_i es el *i*-ésimo atributo de *r*, si el *i*-ésimo atributo no está actualizado, o, si hay que actualizar el atributo, una expresión, que sólo implica constantes y los atributos de *r*, que da el nuevo valor del atributo.

Si se desea seleccionar varias tuplas de *r* y sólo actualizar esas mismas tuplas, se puede utilizar la expresión siguiente, donde *P* denota la condición de selección que escoge las tuplas que hay que actualizar:

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(\sigma_P(r)) \cup (r - \sigma_P(r))$$

Para ilustrar el uso de la operación actualización supóngase que se realiza el pago de los intereses y que hay que aumentar todos los saldos en un 5 por ciento. Hay que escribir

$$cuenta \leftarrow \Pi_{nombre\text{-}sucursal,\ número\text{-}cuenta,\ saldo,\ saldo * 1.05}(cuenta)$$

Supóngase ahora que las cuentas con saldos superiores a 10.000 € reciben un interés del 6 por ciento, mientras que los demás reciben un 5 por ciento. Hay que escribir

$$cuenta \leftarrow \Pi_{NS,\ NC,\ saldo * 1.06}(\sigma_{saldo > 10000}(cuenta)) \cup$$
$$cuenta \leftarrow \Pi_{NS,\ NC,\ saldo * 1.05}(\sigma_{saldo \leq 10000}(cuenta))$$

donde las abreviaturas *NS* y *NC* sustituyen a *nombre-sucursal* y a *número-cuenta*, respectivamente.

3.5. VISTAS

En los ejemplos propuestos hasta ahora se ha operado en el nivel del modelo lógico. Es decir, se ha asumido que el conjunto de relaciones que se da son las relaciones reales guardadas en la base de datos.

No es deseable que todos los usuarios puedan ver la totalidad del modelo lógico. Las consideraciones sobre la seguridad pueden exigir que algunos datos queden ocultos para los usuarios. Considérese una persona que necesita saber el número de préstamo de un cliente pero que no necesita ver el importe del préstamo. Esta persona debería ver una relación descrita en el álgebra relacional mediante

$$\Pi_{nombre\text{-}cliente,\ número\text{-}préstamo,\ nombre\text{-}sucursal}\ (prestatario \bowtie préstamo)$$

Aparte de las consideraciones sobre la seguridad puede que se desee crear un conjunto personalizado de relaciones que se adapte mejor que el modelo lógico a la intuición de un usuario concreto. Por ejemplo, puede que un empleado del departamento de publicidad quiera ver una relación que conste de los clientes que tengan abierta una cuenta o concedido un préstamo en el banco y de las sucursales con las que trabajan. La relación que se crearía para ese empleado es

$$\Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente}\ (impositor \bowtie cuenta) \cup \Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente}\ (prestatario \bowtie préstamo)$$

Las relaciones que no forman parte del modelo lógico pero se hacen visibles a los usuarios como relaciones virtuales se denominan **vistas**. Se puede trabajar con gran número de vistas sobre cualquier conjunto dado de relaciones reales.

3.5.1. Definición de vistas

Las vistas se definen utilizando la instrucción **create view**. Para definir una vista hay que darle un nombre e indicar la consulta que la va a calcular. La forma de la instrucción **create view** es

create view *v* **as** <expresión de consulta>

donde <expresión de consulta> es cualquier expresión legal de consulta del álgebra relacional. El nombre de la vista se representa mediante *v*.

Como ejemplo considérese la vista consistente en las sucursales y sus clientes. Supóngase que se desea que esta vista se denomine *todos-los-clientes*. Esta vista se define de la manera siguiente:

create view *todos-los-clientes* **as**
$$\Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente}\ (impositor \bowtie cuenta) \cup \Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente}\ (prestatario \bowtie préstamo)$$

Una vez se ha definido una vista se puede utilizar el nombre de la vista para hacer referencia a la relación virtual que genera la vista. Utilizando la vista *todos-los-clientes* se puede averiguar el nombre de todos los clientes de la sucursal de Navacerrada escribiendo

$$\Pi_{nombre\text{-}cliente}\ (\sigma_{nombre\text{-}sucursal\ =\ «Navacerrada»}\ (todos\text{-}los\text{-}clientes))$$

Recuérdese que en el Apartado 3.2.1 se escribió la misma consulta sin utilizar vistas.

Los nombres de las vistas pueden aparecer en cualquier lugar en el que pueda encontrarse el nombre de una relación, siempre y cuando no se ejecuten sobre las vistas operaciones de actualización. El asunto de las operaciones de actualización de las vistas se estudia en el Apartado 3.5.2.

La definición de las vistas se diferencia de la operación asignación del álgebra relacional. Supóngase que se define la relación $r1$ de la manera siguiente:

$$r1 \leftarrow \Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente}\ (impositor \bowtie cuenta) \cup \Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente}\ (prestatario \bowtie préstamo)$$

La operación asignación se evalúa una vez, y $r1$ no cambiará cuando se actualicen las relaciones *impositor*, *cuenta*, *préstamo* o *prestatario*. En cambio, si hay alguna modificación en estas relaciones, el conjunto de tuplas de la vista *todos-los-clientes* también cambia. De manera intuitiva, en cualquier momento dado, el conjunto de tuplas de la relación de vistas se define como el resultado de la evaluación de la expresión de consulta que define en ese momento la vista.

Por tanto, si una relación de vistas se calcula y se guarda, puede quedar desfasada si las relaciones utilizadas para definirla se modifican. En vez de eso, las vistas suelen implementarse de la manera siguiente. Cuando se define una vista, el sistema de la base de datos guarda la definición de la propia vista, en vez del resultado de la evaluación de la expresión del álgebra relacional que la define. Siempre que se utiliza una relación de vistas en una consulta, se sustituye por la expresión de consulta guardada. Por tanto, la relación de vistas vuelve a calcularse siempre que se evalúa la consulta.

Algunos sistemas de bases de datos permiten que se guarden las relaciones de vistas, pero se aseguran de que, si las relaciones reales utilizadas en la definición de la vista cambian, la vista se mantenga actualizada. Estas vistas se denominan **vistas materializadas**. El proceso de mantener actualizada la vista se denomina **mantenimiento de vistas**, que se trata en el Apartado 14.5. Las aplicaciones en las que se utiliza frecuentemente una vista se benefician del uso de vistas materializadas, igual que las aplicaciones que demandan una rápida respuesta a ciertas consultas basadas en las vistas. Las ventajas de

la materialización de una vista para las consultas deben sopesarse frente a los costes de almacenamiento y la sobrecarga añadida por las actualizaciones.

3.5.2. Actualizaciones mediante vistas y valores nulos

Aunque las vistas son una herramienta útil para las consultas, plantean problemas significativos si con ellas se expresan las actualizaciones, las inserciones o los borrados. La dificultad radica en que las modificaciones de la base de datos expresadas en términos de vistas deben traducirse en modificaciones de las relaciones reales en el modelo lógico de la base de datos.

Para ilustrar el problema considérese un empleado que necesita ver todos los datos de préstamos de la relación *préstamo* salvo *importe*. Sea *préstamo-sucursal* la vista dada al empleado. Se define esta vista como

create view *préstamo-sucursal* **as**
$\Pi_{nombre\text{-}sucursal,\ número\text{-}préstamo}\ (préstamo)$

Dado que se permite que los nombres de las vistas aparezcan en cualquier parte en la que estén permitidos los nombres de relaciones, el empleado puede escribir:

préstamo-sucursal ← *préstamo-sucursal*
∪ {(P-37, «Navacerrada»)}

Esta inserción debe representarse mediante una inserción en la relación *préstamo*, dado que *préstamo* es la relación real a partir de la cual se genera la vista *préstamo-sucursal*. Sin embargo, para insertar una tupla en *préstamo* hay que tener algún valor para *importe*. Hay dos enfoques razonables para trabajar con esta inserción:

- Rechazar la inserción y devolver al usuario un mensaje de error.
- Insertar una tupla (P-37, «Navacerrada», *nulo*) en la relación *préstamo*.

Otro problema resultante de la modificación de la base de datos mediante las vistas aparece en una vista como la siguiente:

create view *información-crédito* **as**
$\Pi_{nombre\text{-}cliente,\ importe}(prestatario \bowtie préstamo)$

Esta vista da una lista del importe de cada préstamo que tenga concedido cualquier cliente del banco. Considérese la inserción siguiente realizada mediante esta vista:

información-crédito ← *información-crédito*
∪ {(«González», 1900)}

El único método posible de insertar tuplas en las relaciones *prestatario* y *préstamo* es insertar («González», *nulo*) en *prestatario* y (*nulo*, *nulo*, 1900) en *préstamo*. Así, se obtienen las relaciones mostradas en la Figura 3.36. Sin embargo, esta actualización no tiene el efecto deseado, dado que la relación de vistas *información-crédito* sigue *sin* incluir la tupla («González», 1900). Por tanto, no existe manera de actualizar las relaciones *prestatario* y *préstamo* utilizando valores nulos para obtener la actualización deseada de *información-crédito*.

Debido a este tipo de problemas generalmente no se permiten las modificaciones en las relaciones de vistas excepto en casos limitados. Los diferentes sistemas de bases de datos especifican diferentes condiciones bajo las que se permiten actualizaciones sobre las vistas; véanse los manuales de cada sistema de bases de datos en particular para consultar más detalles. El problema general de la modificación de las bases de datos mediante las vistas ha sido objeto de numerosas investigaciones. Las notas bibliográficas hacen mención de trabajos recientes sobre este asunto.

3.5.3. Vistas definidas utilizando otras vistas

En el Apartado 3.5.1 se mencionó que las relaciones de vistas pueden aparecer en cualquier lugar en que pueda hacerlo el nombre de una relación, salvo las restricciones en el uso de vistas en expresiones para la actualización. Por tanto, se pueden utilizar vistas en la expresión que define otra vista. Por ejemplo, se puede definir la vista *cliente-navacerrada* de la manera siguiente:

create view *cliente-navacerrada* **as**
$\Pi_{nombre\text{-}cliente}\,(\sigma_{nombre\text{-}sucursal\ =\ «Navacerrada»}$
$(todos\text{-}los\text{-}clientes))$

donde *todos-los-clientes* es, a su vez, una relación de vistas.

número-préstamo	*nombre-sucursal*	*importe*
P-11	Collado Mediano	900
P-14	Centro	1.500
P-15	Navacerrada	1.500
P-16	Navacerrada	1.300
P-17	Centro	1.000
P-23	Moralzarzal	2.000
P-93	Becerril	500
nulo	*nulo*	1.900

nombre-cliente	*número-préstamo*
Fernández	P-16
Gómez	P-23
Gómez	P-11
López	P-15
Pérez	P-93
Santos	P-17
Sotoca	P-14
Valdivieso	P-17
González	nulo

FIGURA 3.36. Tuplas insertadas en *préstamo* y en *prestatario*.

La **expansión de vistas** es una manera de definir el significado de las vistas definidas en términos de otras vistas. El procedimiento asume que las definiciones de vistas no son **recursivas**; es decir, ninguna vista se usa en su propia definición, bien directa o indirectamente a través de otras definiciones de vistas. Por ejemplo, si v_1 se usa en la definición de v_2, se usa en la definición de v_3, y v_3 se usa en la definición de v_1, entonces v_1, v_2 y v_3 son recursivas. Las definiciones de vistas recursivas son útiles en algunos casos, y se volverá a ellas en el contexto del lenguaje Datalog, en el Apartado 5.2.

Sea la vista v_1 definida mediante una expresión e_1 que puede contener a su vez relaciones de vistas. Las relaciones de vistas representan a las expresiones que definen las vistas y, por tanto, se pueden sustituir por las expresiones que las definen. Si se modifica una expresión sustituyendo una relación de vistas por su definición, la expresión resultante puede seguir conteniendo otras relaciones de vistas. Por tanto, la expansión de vistas de una expresión repite la etapa de sustitución de la manera siguiente:

repeat
  Buscar todas las relaciones de vistas v_i de e_1
  Sustituir la relación de vistas v_i por la expresión que define v_i
until no queden más relaciones de vistas en e_1

Mientras las definiciones de las vistas no sean recursivas el bucle concluirá. Por tanto, una expresión *e* que contenga relaciones de vistas puede entenderse como la expresión resultante de la expansión de vistas de *e*, que no contiene ninguna relación de vistas.

Como ilustración de la expansión de vistas considérese la expresión siguiente:

$$\sigma_{nombre\text{-}cliente = \text{«Martín»}}(cliente\text{-}navacerrada)$$

El procedimiento de expansión de vistas produce inicialmente

$$\sigma_{nombre\text{-}cliente = \text{«Martín»}}(\Pi_{nombre\text{-}cliente}(\sigma_{nombre\text{-}sucursal = \text{«Navacerrada»}}(todos\text{-}los\text{-}clientes)))$$

luego produce

$$\sigma_{nombre\text{-}cliente = \text{«Martín»}}(\Pi_{nombre\text{-}cliente}(\sigma_{nombre\text{-}sucursal = \text{«Navacerrada»}}(\Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente}(impositor \bowtie cuenta) \cup \Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente}(prestatario \bowtie préstamo))))$$

No hay más usos de las relaciones de vistas y concluye la expansión de vistas.

3.6. EL CÁLCULO RELACIONAL DE TUPLAS

Cuando escribimos una expresión del álgebra relacional proporcionamos una serie de procedimientos que generan la respuesta a la consulta. El cálculo relacional de tuplas, en cambio, es un lenguaje de consulta **no procedimental**. Describe la información deseada sin dar un procedimiento específico para obtenerla.

Las consultas se expresan en el cálculo relacional de tuplas como

$$\{t \mid P(t)\}$$

es decir, son el conjunto de todas las tuplas tales que el predicado *P* es cierto para *t*. Siguiendo la notación utilizada previamente, se utiliza $t[A]$ para denotar el valor de la tupla *t* en el atributo *A* y $t \in r$ para denotar que la tupla *t* está en la relación *r*.

Antes de dar una definición formal del cálculo relacional de tuplas se volverán a examinar algunas de las consultas para las que se escribieron expresiones de álgebra relacional en el Apartado 3.2.

3.6.1. Consultas de ejemplo

Supóngase que se desea averiguar *nombre-sucursal*, *número-préstamo* e *importe* de los préstamos superiores a 1.200 €:

$$\{t \mid t \in préstamo \wedge t[importe] > 1200\}$$

Supóngase que sólo se desea obtener el atributo *número-préstamo*, en vez de todos los atributos de la relación *préstamo*. Para escribir esta consulta en el cálculo relacional de tuplas hay que escribir una expresión para una relación del esquema (*número-préstamo*). Se necesitan las tuplas de (*número-préstamo*) tales que hay una tupla en *préstamo* con el atributo *importe* > 1200. Para expresar esta solicitud hay que utilizar el constructor «existe» de la lógica matemática. La notación

$$\exists\, t \in r\ (Q\ (t))$$

significa «existe una tupla *t* en la relación *r* tal que el predicado $Q(t)$ es verdadero».

Utilizando esta notación se puede escribir la consulta «Averiguar el número de préstamo de todos los préstamos por importe superior a 1.200 €» como

$$\{t \mid \exists\, s \in préstamo\ (t[número\text{-}préstamo] = s[número\text{-}préstamo] \wedge s[importe] > 1200)\}$$

En español la expresión anterior se lee «el conjunto de todas las tuplas *t* tales que existe una tupla *s* en la relación *préstamo* para la que los valores de *t* y de *s* para el

atributo *número-préstamo* son iguales y el valor de *s* para el atributo *importe* es mayor que 1.200 €».

La variable tupla *t* sólo se define para el atributo *número-préstamo*, dado que es el único atributo para el que se especifica una condición para *t*. Por tanto, el resultado es una relación de (*número-préstamo*).

Considérese la consulta «Averiguar el nombre de todos los clientes que tienen concedido un préstamo en la sucursal de Navacerrada». Esta consulta es un poco más compleja que las anteriores, dado que implica a dos relaciones: *prestatario* y *préstamo*. Como se verá, sin embargo, todo lo que necesita es que tengamos dos instrucciones «existe» en la expresión de cálculo relacional de tuplas, relacionadas mediante *y* (∧). La consulta se escribe de la manera siguiente:

$$\{t \mid \exists\, s \in prestatario\ (t[número\text{-}préstamo] = s[número\text{-}préstamo] \wedge \exists\, u \in préstamo\ (u[número\text{-}préstamo] = s[número\text{-}préstamo] \wedge u[nombre\text{-}sucursal] = \text{«Navacerrada»}))\}$$

En español esta expresión es «el conjunto de todas las tuplas (*nombre-cliente*) para las que el cliente tiene un préstamo concedido en la sucursal de Navacerrada». La variable tupla *u* asegura que el cliente es prestatario de la sucursal de Navacerrada. La variable tupla *s* está restringida para que corresponda al mismo número de préstamo que *s*. El resultado de esta consulta se muestra en la Figura 3.37.

Para averiguar todos los clientes del banco que tienen concedido un préstamo, una cuenta abierta, o ambas cosas, se utilizó la operación unión del álgebra relacional. En el cálculo relacional de tuplas harán falta dos instrucciones «existe» relacionadas por *o* (∨):

$$\{t \mid \exists\, s \in prestatario\ (t[nombre\text{-}cliente] = s[nombre\text{-}cliente]) \vee \exists\, u \in impositor\ (t[nombre\text{-}cliente] = u[nombre\text{-}cliente])\}$$

Esta expresión da el conjunto de todas las tuplas de *nombre-cliente* tales que se cumple al menos una de las condiciones siguientes:

- *nombre-cliente* aparece en alguna tupla de la relación *prestatario* como prestatario del banco.
- *nombre-cliente* aparece en alguna tupla de la relación *impositor* como impositor del banco.

Si algún cliente tiene concedido un préstamo y una cuenta abierta en el banco, ese cliente sólo aparece una vez en el resultado, ya que la definición matemática de conjunto no permite elementos duplicados. El resultado de esta consulta se mostró previamente en la Figura 3.12.

Si *sólo* queremos conocer los clientes que tienen en el banco una cuenta *y* un préstamo, todo lo que hay que hacer es cambiar en la expresión anterior la *o* (∨) por una *y* (∧).

$$\{t \mid \exists\, s \in prestatario\ (t[nombre\text{-}cliente] = s[nombre\text{-}cliente]) \wedge \exists\, u \in impositor\ (t[nombre\text{-}cliente] = u[nombre\text{-}cliente])\}$$

El resultado de esta consulta se mostró en la Figura 3.20.

Considérese ahora la consulta «Averiguar todos los clientes que tienen una cuenta abierta en el banco pero no tienen concedido ningún préstamo». La expresión del cálculo relacional de tuplas para esta consulta es parecida a las expresiones que se acaban de ver, salvo el uso del símbolo *no* (¬):

$$\{t \mid \exists\, u \in impositor\ (t[nombre\text{-}cliente] = u[nombre\text{-}cliente]) \wedge \neg \exists\, s \in prestatario\ (t[nombre\text{-}cliente] = s[nombre\text{-}cliente])\}$$

La expresión del cálculo relacional de tuplas anterior utiliza la instrucción $\exists\, u \in impositor\ (\ldots)$ para exigir que el cliente tenga una cuenta abierta en el banco, y utiliza la instrucción $\neg \exists\, s \in prestatario\ (\ldots)$ para borrar a aquellos clientes que aparecen en alguna tupla de la relación *prestatario* por tener un préstamo del banco. El resultado de esta consulta apareció en la Figura 3.13.

La consulta que se tomará ahora en consideración utiliza la implicación, denotada por ⇒. La fórmula $P \Rightarrow Q$ es lógicamente equivalente a $\neg P \vee Q$. El uso de la implicación en lugar de *no* y *o* suele sugerir una interpretación más intuitiva de la consulta en español.

Considérese la consulta que se utilizó en el Apartado 3.2.3 para ilustrar la operación división: «Averiguar todos los clientes que tienen una cuenta en todas las sucursales sitas en Arganzuela». Para escribir esta consulta en el cálculo relacional de tuplas se introduce el constructor «para todo», denotado por ∀. La notación

$$\forall\, t \in r\ (Q\ (t))$$

significa «*Q* es verdadera para todas las tuplas *t* de la relación *r*».

La expresión para la consulta se escribe de la manera siguiente:

$$\{t \mid \exists\, r \in cliente\ (r[nombre\text{-}cliente] = t[nombre\text{-}cliente] \wedge (\forall\, u \in sucursal\ (u[ciudad\text{-}sucursal] = \text{«Arganzuela»} \Rightarrow \exists\, s \in impositor\ (t[nombre\text{-}cliente] = s[nombre\text{-}cliente] \wedge \exists\, w \in cuenta\ (w[número\text{-}cuenta] = s[número\text{-}cuenta] \wedge w[nombre\text{-}sucursal] = u[nombre\text{-}sucursal])))))\}$$

nombre-cliente
Fernández
López

FIGURA 3.37. Nombre de todos los clientes que tienen concedido un préstamo en la sucursal de Navacerrada.

En español esta expresión se interpreta como «el conjunto de todos los clientes (es decir, las tuplas t (*nombre-cliente*)) tales que, para todas las tuplas u de la relación *sucursal*, si el valor de u en el atributo *ciudad-sucursal* es Arganzuela, el cliente tiene una cuenta en la sucursal cuyo nombre aparece en el atributo *nombre-sucursal* de u».

Nótese que hay una sutileza en la consulta anterior: si no hay ninguna sucursal en Arganzuela, todos los nombres de cliente satisfacen la condición. La primera línea de la expresión de consulta es crítica en este caso: sin la condición

$$\exists\, r \in cliente\ (r[nombre\text{-}cliente] = t[nombre\text{-}cliente]$$

si no hay sucursal en Arganzuela, cualquier valor de t (incluyendo los valores que no son nombres de cliente en la relación *cliente*) valdría.

3.6.2. Definición formal

Ahora se tiene la preparación necesaria para una definición formal. Las expresiones del cálculo relacional de tuplas son de la forma

$$\{t \mid P\,(t)\}$$

donde P es una *fórmula*. En una fórmula pueden aparecer varias variables tupla. Se dice que una variable tupla es una *variable libre* a menos que esté cuantificada mediante $\exists$ o $\forall$. Por tanto, en

$$t \in préstamo \wedge \exists\, s \in cliente\ (t[nombre\text{-}sucursal] = s[nombre\text{-}sucursal])$$

t es una variable libre. La variable tupla s se denomina variable *ligada*.

Las fórmulas de cálculo relacional de tuplas se construyen con *átomos*. Los átomos tienen una de las formas siguientes:

- $s \in r$, donde s es una variable tupla y r es una relación (no se permite el uso del operador $\notin$)
- $s[x]\ \Theta\ u[y]$, donde s y u son variables tuplas, x es un atributo en el que está definida s, y es un atributo en el que está definida u y Θ es un operador de comparación ($<, \leq, =, \neq, >, \geq$); es necesario que los atributos x e y tengan dominios cuyos miembros puedan compararse mediante Θ
- $s[x]\ \Theta\ c$, donde s es una variable tupla, x es un atributo en el que está definida s, Θ es un operador de comparación y c es una constante en el dominio del atributo x

Las fórmulas se construyen a partir de los átomos utilizando las reglas siguientes:

- Un átomo es una fórmula.
- Si P_1 es una fórmula, también lo son $\neg\, P_1$ y (P_1).
- Si P_1 y P_2 son fórmulas, también lo son $P_1 \vee P_2$, $P_1 \wedge P_2$ y $P_1 \Rightarrow P_2$.
- Si $P_1(s)$ es una fórmula que contiene una variable tupla libre s, y r es una relación,

$$\exists\, s \in r\ (P_1\,(s)) \ \text{ y } \ \forall\, s \in r\ (P_1\,(s))$$

también son fórmulas

Igual que en el álgebra relacional, se pueden escribir expresiones equivalentes que no sean idénticas en apariencia. En el cálculo relacional de tuplas estas equivalencias incluyen las tres reglas siguientes:

1. $P_1 \wedge P_2$ es equivalente a $\neg\,(\neg\,(P_1) \vee \neg\,(P_2))$.
2. $\forall\, t \in r\ (P_1\,(t))$ es equivalente a $\neg\,\exists\, t \in r\ (\neg\, P_1\,(t))$.
3. $P_1 \Rightarrow P_2$ es equivalente a $\neg\,(P_1) \vee P_2$.

3.6.3. Seguridad de las expresiones

Queda un último asunto por tratar. Las expresiones del cálculo relacional de tuplas pueden generar relaciones infinitas. Supóngase que se escribió la expresión

$$\{t \mid \neg\,(t \in préstamo)\}$$

Hay infinitas tuplas que no están en *préstamo*. La mayor parte de estas tuplas contienen valores que ni siquiera aparecen en la base de datos. Resulta evidente que no se desea permitir ese tipo de expresiones.

Para ayudar a definir las restricciones del cálculo relacional de tuplas se introduce el concepto de **dominio** de una fórmula relacional de tuplas, P. De manera intuitiva, el dominio de P, denotado por $dom(P)$, es el conjunto de todos los valores a los que P hace referencia. Esto incluye a los valores mencionados en la propia P, así como a los valores que aparezcan explícitamente en P o en una o en varias relaciones cuyos nombres aparezcan en P. Así, el dominio de P es el conjunto de todos los valores que aparecen explícitamente en una o más relación cuyos nombres aparecen en P. Por ejemplo, $dom(t \in préstamo \wedge t[importe] > 1200)$ es el conjunto que contiene a 1200 y el conjunto de todos los valores que aparecen en *préstamo*. Además, $dom(\neg\,(t \in préstamo))$ es el conjunto de todos los valores que aparecen en *préstamo*, dado que la relación *préstamo* se menciona en la expresión.

Se dice que una expresión $\{t \mid P(t)\}$ es *segura* si todos los valores que aparecen en el resultado son valores de $dom(P)$. La expresión $\{t \mid \neg\,(t \in préstamo)\}$ no es segura. Obsérvese que $dom(\neg\,(t \in préstamo))$ es el conjunto de todos los valores que aparecen en *préstamo*. Sin embargo, es posible tener una tupla t que no esté en *préstamo* que contenga valores que no aparezcan en *préstamo*. El resto de ejemplos de expresiones del cálculo relacional de tuplas que se han escrito en este apartado son seguros.

3.6.4. Potencia expresiva de los lenguajes

El cálculo relacional de tuplas restringido a expresiones seguras es equivalente en potencia expresiva al álgebra relacional básica (con los operadores ∪, – , ×, σ y ρ, pero sin los operadores relacionales extendidos tales como la proyección generalizada *G* y las operaciones de reunión externa). Por tanto, para cada expresión del álgebra relacional hay una expresión equivalente del cálculo relacional de tuplas, y para cada expresión del cálculo relacional de tuplas hay una expresión equivalente del álgebra relacional. No se probará aquí esta afirmación; las notas bibliográficas contienen referencias a la demostración. Algunas partes de la misma se incluyen en los ejercicios. El cálculo relacional de tuplas no tiene ningún equivalente de la operación agregación, pero se puede extender para contenerla. La extensión del cálculo relacional de tuplas para manejar las expresiones aritméticas es sencilla.

3.7. EL CÁLCULO RELACIONAL DE DOMINIOS**

Hay una segunda forma de cálculo relacional denominada **cálculo relacional de dominios**. Esta forma utiliza variables de *dominio* que toman sus valores del dominio de un atributo, en vez de tomarlos de una tupla completa. El cálculo relacional de dominios, sin embargo, se halla estrechamente relacionado con el cálculo relacional de tuplas.

3.7.1. Definición formal

Las expresiones del cálculo relacional de dominios son de la forma

$$\{< x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

donde $x_1, x_2, \ldots, x_n$ representan las variables de dominio, *P* representa una fórmula compuesta de átomos, como era el caso en el cálculo relacional de tuplas. Los átomos del cálculo relacional de dominios tienen una de las formas siguientes:

- $< x_1, x_2, \ldots, x_n > \in r$, donde *r* es una relación con *n* atributos y $x_1, x_2, \ldots, x_n$ son variables de dominio o constantes de dominio.
- $x \; \Theta \; y$, donde *x* e *y* son variables de dominio y Θ es un operador de comparación (<, ≤, =, ≠, >, ≥). Se exige que los atributos *x* e *y* tengan dominios que puedan compararse mediante Θ.
- $x \; \Theta \; c$, donde *x* es una variable de dominio, Θ es un operador de comparación y *c* es una constante del dominio del atributo para el que *x* es una variable de dominio.

Las fórmulas se construyen a partir de los átomos utilizando las reglas siguientes:

- Un átomo es una fórmula.
- Si P_1 es una fórmula, también lo son $\neg P_1$ y (P_1).
- Si P_1 y P_2 son fórmulas, también lo son $P_1 \lor P_2$, $P_1 \land P_2$ y $P_1 \Rightarrow P_2$.
- Si $P_1(x)$ es una fórmula en *x*, donde *x* es una variable de dominio,

$$\exists \, x < (P_1(x)) \;\; \text{y} \;\; \forall \, x \, (P_1(x))$$

también son fórmulas

Como notación abreviada se escribe

$$\exists \, a, b, c \, (P(a, b, c))$$

en lugar de

$$\exists \, a \, (\exists \, b \, (\exists \, c \, (P(a, b, c))))$$

3.7.2. Consultas de ejemplo

Ahora se van a aportar consultas del cálculo relacional de dominios para los ejemplos considerados anteriormente. Obsérvese la similitud de estas expresiones con las expresiones correspondientes del cálculo relacional de tuplas

- Averiguar el nombre de la sucursal, el número de préstamo y el importe de los préstamos superiores a 1.200 €:

$$\{< p, s, i > \mid < p, s, i > \in préstamo \land i > 1200\}$$

- Averiguar todos los números de préstamo de los préstamos por importe superior a 1.200 €:

$$\{< p > \mid \exists \, s, i \, (< p, s, i > \in préstamo \land i > 1200)\}$$

Aunque la segunda consulta tenga un aspecto muy parecido al de la que se escribió para el cálculo relacional de tuplas, hay una diferencia importante. En el cálculo de tuplas, cuando se escribe ∃ *s* para alguna variable tupla *s*, se vincula inmediatamente con una relación escribiendo $\exists \, s \in r$. Sin embargo, cuando se escribe ∃ *s* en el cálculo de dominios, *s* no se refiere a una tupla, sino a un valor de dominio. Por tanto, el dominio de la variable *s* no está restringido hasta que la subfórmula *p, s, i* ∈ *préstamo* restringe *s* a los nom-

bres de sucursal que aparecen en la relación *préstamo*. Por ejemplo:

- Averiguar el nombre de todos los clientes que tienen concedido un préstamo en la sucursal de Navacerrada y averiguar el importe del préstamo:

  $\{< n, c > \mid \exists\, l\ (< n, p > \in prestatario \wedge \exists\, s\ (< p, s, i > \in préstamo \wedge s =$ «Navacerrada»$))\}$

- Averiguar el nombre de todos los clientes que tienen concedido un préstamo, una cuenta abierta, o ambas cosas, en la sucursal de Navacerrada:

  $\{< n > \mid \exists\, p\ (< n, p > \in prestatario \wedge \exists\, s, i\ (< p, s, i > \in préstamo \wedge s =$ «Navacerrada»$))$ $\vee \exists\, c\ (< n, c > \in impositor \wedge \exists\, s, i\ (< c, s, i > \in cuenta \wedge s =$ «Navacerrada»$))\}$

- Averiguar el nombre de todos los clientes que tienen una cuenta abierta en todas las sucursales sitas en Arganzuela:

  $\{< c > \mid \exists\, s, t\ (< c, s, t > \in cliente) \wedge \forall\, x, y, z\ (< x, y, z > \in sucursal) \wedge y =$ «Arganzuela» $\Rightarrow \exists\, a, b\ (< x, a, b > \in cuenta \wedge (< c, a > \in impositor))\}$

  En español la expresión anterior se interpreta como «el conjunto de todas las tuplas *c* (*nombre-cliente*) tales que, para todas las tuplas *x*, *y*, *z* (*nombre-sucursal*, *ciudad-sucursal*, *activos*), si la ciudad de la sucursal es Arganzuela, las siguientes afirmaciones son verdaderas»:

  — Existe una tupla de la relación *cuenta* con número de cuenta *a* y nombre de sucursal *x*

  — Existe una tupla de la relación *impositor* con cliente *c* y número de cuenta *a*

3.7.3. Seguridad de las expresiones

Ya se observó que en el cálculo relacional de tuplas es posible escribir expresiones que pueden generar relaciones infinitas. Esto llevó a definir la *seguridad* de las expresiones de cálculo relacional de tuplas. Se produce una situación parecida en el cálculo relacional de dominios. Las expresiones como

$$\{< p, s, i > \mid \neg\ (< p, s, i > \in préstamo)\}$$

no son seguras porque permiten valores del resultado que no están en el dominio de la expresión.

En el cálculo relacional de dominios también hay que tener en cuenta la forma de las fórmulas dentro de las instrucciones «existe» y «para todo». Considérese la expresión

$$\{< x > \mid \exists\, y\ (< x, y \in r) \wedge \exists\, z\ (\neg\ (< x, z > \in r) \wedge P(x, z))\}$$

donde *P* es una fórmula que implica a *x* y a *z*. Se puede probar la primera parte de la fórmula, $\exists\, y\ (< x, y \in r)$, tomando en consideración sólo los valores de *r*. Sin embargo, para probar la segunda parte de la fórmula, $\exists\, z\ (\neg\ (< x, z > \in r) \wedge P(x, z))$, hay que tomar en consideración valores de *z* que no aparecen en *r*. Dado que todas las relaciones son finitas, no aparece en *r* un número infinito de valores. Por tanto, no resulta posible en general probar la segunda parte de la fórmula $\exists\, z\ (\neg\ (< x, z > \in r) \wedge P(x, z))$, hay que tomar en consideración valores de *z* que no aparecen en *r*. Dado que todas las relaciones son finitas, no aparece en *r* un número infinito de valores. Por tanto, no es posible en general probar la segunda parte de la fórmula sin tomar en consideración un número infinito de valores de *z*. En vez de eso, se añaden restricciones para prohibir expresiones como la anterior.

En el cálculo relacional de tuplas se restringió cualquier variable cuantificada existencialmente a variar sobre una relación concreta. Dado que no se hizo así en el cálculo de dominios, hay que añadir reglas a la definición de seguridad para tratar los casos parecidos a los del ejemplo. Se dice que la expresión

$$\{< x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

es segura si se cumplen todas las condiciones siguientes:

1. Todos los valores que aparecen en las tuplas de la expresión son valores de $dom(P)$.
2. Para cada subfórmula «existe» de la forma $\exists\, x\ (P_1(x))$, la subfórmula es cierta si y sólo si hay un valor *x* en $dom(P_1)$ tal que $P_1(x)$ es verdadero.
3. Para cada subfórmula «para todo» de la forma $\forall\, x\ (P_1(x))$, la subfórmula es verdadera si y sólo si $P_1(x)$ es verdadero para todos los valores *x* de $dom(P_1)$.

El propósito de las reglas adicionales es asegurar que se puedan probar las subfórmulas «para todo» y «existe» sin tener que probar infinitas posibilidades. Considérese la segunda regla de la definición de seguridad. Para que $\exists\, x\ (P_1(x))$ sea verdadero sólo hay que encontrar una *x* para la que $P_1\ (x)$ lo sea. En general, habría que probar infinitos valores. Sin embargo, si la expresión es segura, se sabe que se puede restringir la atención a los valores de $dom(P_1)$. Esta restricción reduce las tuplas que hay que tomar en consideración a un número finito.

La situación de las subfórmulas de la forma $\forall\, x\ (P_1(x))$ es parecida. Para asegurar que $\forall\, x\ (P_1(x))$ es verdadero hay que probar en general todos los valores posibles, por lo que hay que examinar infinitos valores. Como antes, si se sabe que la expresión es segura, basta con probar $P_1(x)$ para los valores tomados de $dom(P_1)$.

Todas las expresiones del cálculo relacional de dominios que se han incluido en las consultas de ejemplo de este apartado son seguras.

3.7.4. Potencia expresiva de los lenguajes

Cuando el cálculo relacional de dominios se restringe a expresiones seguras es equivalente en potencia expresiva al cálculo relacional de tuplas restringido a expresiones seguras. Dado que se observó anteriormente que el cálculo relacional de tuplas restringido es equivalente al álgebra relacional, los tres lenguajes siguientes son equivalentes:

- El álgebra relacional básica (sin las operaciones extendidas)
- El cálculo relacional de tuplas restringido a expresiones seguras
- El cálculo relacional de dominios restringido a expresiones seguras

El cálculo relacional de dominios tampoco tiene equivalente para la operación agregación, pero se puede extender para contenerla, y su extensión para el tratamiento de expresiones aritméticas es sencilla.

3.8. RESUMEN

- El **modelo de datos relacional** se basa en un conjunto de tablas. El usuario del sistema de bases de datos puede consultar esas tablas, insertar nuevas tuplas, borrar tuplas y actualizar (modificar) las tuplas. Hay varios lenguajes para expresar estas operaciones.
- El **álgebra relacional** define un conjunto de operaciones algebraicas que operan sobre tablas y devuelven tablas como resultado. Estas operaciones se pueden combinar para obtener expresiones que expresan las consultas deseadas. El álgebra define las operaciones básicas usadas en los lenguajes de consulta relacionales.
- Las operaciones del álgebra relacional se pueden dividir en :
  - — Operaciones básicas
  - — Operaciones adicionales que se pueden expresar en términos de las operaciones básicas
  - — Operaciones extendidas, algunas de las cuales añaden mayor poder expresivo al álgebra relacional
- Las bases de datos se pueden modificar con la **inserción**, el **borrado** y la **actualización** de tuplas. Se usó el álgebra relacional con el **operador de asignación** para expresar estas modificaciones.
- Los diferentes usuarios de una base de datos compartida pueden aprovecharse de vistas individualizadas de la base de datos. Las vistas son «relaciones virtuales» definidas mediante expresiones de consulta.
- Las vistas son mecanismos útiles para simplificar las consultas a la base de datos, pero la modificación de la base de datos mediante las vistas puede tener consecuencias potencialmente desventajosas. Por tanto, los sistemas de bases de datos restringen estrictamente las actualizaciones mediante vistas.
- Por razones de eficiencia del procesamiento de las consultas, una vista puede estar **materializada**, es decir, la consulta se evalúa y el resultado se almacena físicamente. Cuando las relaciones de la base de datos se actualizan, la vista materializada se debe actualizar correspondientemente.
- El **cálculo relacional de tuplas** y el **cálculo relacional de dominios** son lenguajes no procedimentales que representan la potencia básica necesaria en un lenguaje de consultas relacionales. El álgebra relacional básica es un lenguaje procedimental que es equivalente en potencia a ambas formas del cálculo relacional cuando se restringen a las expresiones seguras.
- El álgebra relacional y los cálculos relacionales son lenguajes rígidos, formales, que no resultan adecuados para los usuarios ocasionales de los sistemas de bases de datos. Los sistemas comerciales de bases de datos, por tanto, utilizan lenguajes con más «azúcar sintáctico». En los Capítulos 4 y 5 se tomarán en consideración los tres lenguajes comerciales más influyentes: **SQL**, que está basado en el álgebra relacional, **QBE** y **Datalog**, que están basados en el cálculo relacional de dominios.

TÉRMINOS DE REPASO

- Agrupación
- Álgebra relacional
- Cálculo relacional de dominios
- Cálculo relacional de tuplas
- Clave externa
  - Relación referenciada
  - Relación referenciante
- Claves
- Definición de vistas
- Dominio atómico
- Diagrama de esquema
- Ejemplar de la base de datos
- Ejemplar de la relación
- Esquema de la base de datos
- Esquema de la relación
- Expansión de vistas
- Lenguaje de consulta
- Lenguaje procedimental
- Lenguaje no procedimental
- Modificación de la base de datos
  - Actualización
  - Borrado
  - Inserción
- Multiconjuntos
- Operaciones adicionales
  - División /
  - Intersección de conjuntos $\cap$
  - Reunión natural $\bowtie$
- Operaciones del álgebra relacional
  - Diferencia de conjuntos –
  - Producto cartesiano $\times$
  - Proyección Π
  - Renombramiento ρ
  - Selección σ
  - Unión $\cup$
- Operaciones del álgebra relacional extendida
  - Agregación G
  - Proyección generalizada Π
  - Reunión externa
    - Reunión externa completa ⟗
    - Reunión externa por la derecha ⟖
    - Reunión externa por la izquierda ⟕
- Operación asignación
- Potencia expresiva de los lenguajes
- Relación
- Seguridad de las expresiones
- Tabla
- Valor nulo
- Valores nulos
- Variable tupla
- Vistas
- Vistas recursivas

EJERCICIOS

3.1. Diséñese una base de datos relacional para la oficina de registro de una universidad. La oficina conserva datos sobre cada curso, incluyendo el profesor, el número de estudiantes matriculados y la hora y el lugar de las clases. Por cada pareja estudiante-curso se guarda una calificación.

3.2. Descríbanse las diferencias de significado entre los términos *relación* y *esquema de la relación*. Ilústrese la respuesta haciendo referencia a la solución propuesta para el Ejercicio 3.1.

3.3. Diséñese una base de datos relacional correspondiente al diagrama E-R de la Figura 3.38.

3.4. En el Capítulo 2 se mostró la manera de representar los conjuntos de relaciones de varios a varios, de varios a uno, de uno a varios y de uno a uno. Explíquese la manera en que las claves primarias ayudan a representar estos conjuntos de relaciones en el modelo relacional.

3.5. Considérese la base de datos relacional de la Figura 3.39. Dese una expresión del álgebra relacional, otra del cálculo relacional de tuplas y una tercera del cálculo relacional de dominios para cada una de las consultas siguientes:

a. Averiguar los nombres de todos los empleados que trabajan para el Banco Importante.

b. Averiguar el nombre y la ciudad de residencia de todos los empleados que trabajan para el Banco Importante.

c. Averiguar el nombre, la calle y la ciudad de residencia de todos los empleados que trabajan para el Banco Importante y ganan más de 2.000.000 de pesetas anuales.

d. Averiguar el nombre de todos los empleados de esta base de datos que viven en la misma ciudad que la compañía para la que trabajan.

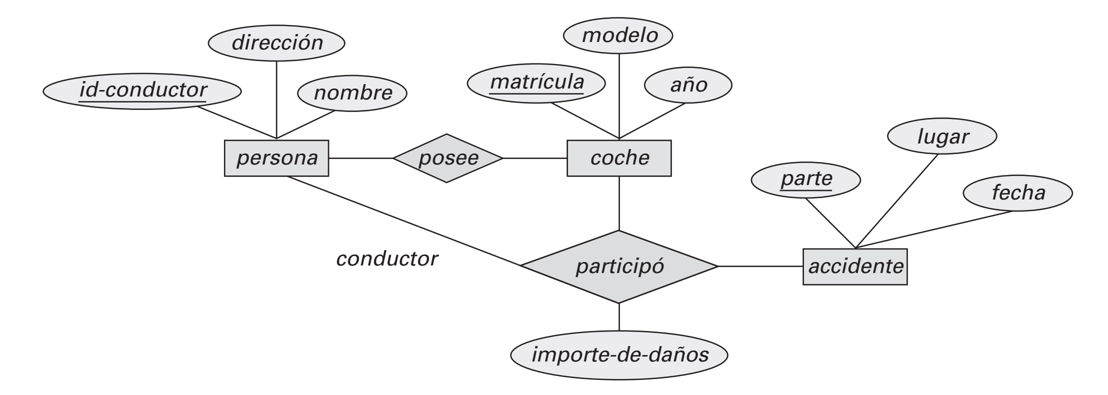


FIGURA 3.38. Diagrama E-R.

e. Averiguar el nombre de todos los empleados que viven en la misma ciudad y en la misma calle que sus jefes.

f. Averiguar el nombre de todos los empleados de esta base de datos que no trabajan para el Banco Importante.

g. Averiguar el nombre de todos los empleados que ganan más que cualquier empleado del Banco Pequeño.

h. Supóngase que las compañías pueden estar instaladas en ciudades pequeñas. Hállense todas las compañías instaladas en cada ciudad en la que está instalado el Banco Pequeño.

3.6. Considérese la relación de la Figura 3.21, que muestra el resultado de la consulta «Averígüese el nombre de todos los clientes que tienen concedido un préstamo en el banco». Vuélvase a escribir la consulta para incluir no sólo el nombre, sino también la ciudad de residencia de cada cliente. Obsérvese que ahora el cliente Sotoca ya no aparece en el resultado, aunque en realidad tiene un préstamo concedido por el banco.

a. Explíquese el motivo de que Sotoca no aparezca en el resultado.

b. Supóngase que se desea que Sotoca aparezca en el resultado. ¿Cómo habría que modificar la base de datos para conseguirlo?

c. Una vez más, supóngase que se desea que Sotoca aparezca en el resultado. Escríbase una consulta utilizando una reunión externa que cumpla esta condición sin que haya que modificar la base de datos.

3.7. Las operaciones de reunión externa amplían la operación reunión natural de manera que las tuplas de las relaciones participantes no se pierdan en el resultado de la reunión. Descríbase la manera en que la operación reunión zeta puede ampliarse para que las tuplas de la relación de la izquierda, las de la relación de la derecha o las de ambas relaciones no se pierdan en el resultado de una reunión zeta.

empleado (*nombre-empleado*, *calle*, *ciudad*)
trabaja (*nombre-empleado*, *nombre-empresa*, *sueldo*)
empresa (*nombre-empresa*, *ciudad*)
jefe (*nombre-empleado*, *nombre-jefe*)

FIGURA 3.39. Base de datos relacional para los Ejercicios 3.5 y 3.10.

3.8. Considérese la base de datos regional de la Figura 3.39. Dese una expresión del álgebra relacional para cada petición:

a. Modificar la base de datos de manera que Santos viva ahora en Tres Cantos.

b. Dar a todos los empleados del Banco Importante un aumento de sueldo del 10 por ciento.

c. Dar a todos los jefes de la base de datos un aumento de sueldo del 10 por ciento.

d. Dar a todos los jefes de la base de datos un aumento de sueldo del 10 por ciento, a menos que el sueldo resultante sea mayor que 100.000 €. En este caso, dar sólo un aumento del 3 por ciento.

e. Borrar todas las tuplas de los empleados de Banco Pequeño de la relación *trabajo*.

3.9. Utilizando el ejemplo bancario, escríbanse consultas del álgebra relacional para averiguar las cuentas abiertas por más de dos clientes:

a. utilizando una función de agregación.

b. sin utilizar funciones de agregación.

3.10. Considérese la base de datos relacional de la Figura 3.38. Dese una expresión del álgebra relacional para cada una de las consultas siguientes:

a. Averiguar la compañía con mayor número de empleados.

b. Averiguar la compañía con la nómina (suma de sueldos de sus empleados) más reducida.

c. Averiguar las compañías cuyos empleados ganen un sueldo más elevado, en media, que el sueldo medio del Banco Importante.

3.11. Dense dos motivos por los que se puede decidir definir una vista.

3.12. Cítense dos problemas importantes del procesamiento de la operación actualización expresadas en términos de vistas.

3.13. Sean los siguientes esquemas de relaciones:

$$R = (A, B, C)$$
$$S = (D, E, F)$$

Sean las relaciones $r(R)$ y $s(S)$. Dese una expresión del cálculo relacional de tuplas que sea equivalente a cada una de las expresiones siguientes:

a. $\Pi_A(R)$
b. $\sigma_{B = 17}(r)$
c. $r \times s$
d. $\Pi_{A, F}(\sigma_{C = D}(r \times s))$

3.14. Sea $R = (A, B, C)$ y sean r_1 y r_2 relaciones del esquema R. Dese una expresión del cálculo relacional de dominios que sea equivalente a las expresiones siguientes:

a. $\Pi_A(r_1)$
b. $\sigma_{B = 17}(r_1)$
c. $r_1 \cup r_2$
d. $r_1 \cap r_2$
e. $r_1 - r_2$
f. $\Pi_{A, B}(r_1) \bowtie \Pi_{B, C}(r_2)$

3.15. Repítase el Ejercicio 3.5 usando el cálculo relacional de tuplas y el de dominios.

3.16. Sean $R = (A, B)$ y $S = (A, C)$ y sean $r(R)$ y $s(S)$ relaciones. Escríbanse expresiones del álgebra relacional equivalentes a las expresiones siguientes del cálculo relacional de dominios:

a. $\{< a > \mid \exists\, b\, (< a, b > \in r \land b = 17)\}$
b. $\{< a, b, c > \mid < a, b > \in r \land < a, c > \in s\}$
c. $\{< a > \mid \exists\, b\, (< a, b > \in r) \lor \forall\, c\, (\exists\, d\, (< d, c > \in s) \Rightarrow < a, c > \in s)\}$
d. $\{< a > \mid \exists\, c\, (< a, c > \in s) \land \exists\, b_1, b_2\, (< a, b_1 > \in r \land < c, b_2 > \in r \land b_1 > b_2))\}$

3.17. Sea $R = (A, B)$ y $S = (A, C)$ y sean $r(R)$ y $s(S)$ relaciones. Utilizando la constante especial *nulo*, escríbanse expresiones del cálculo relacional de tuplas equivalentes a cada una de las expresiones siguientes:

a. $r ⟖ s$
b. $r ⟗ s$
c. $r ⟕ s$

3.18. Dense dos motivos por los que se puedan introducir valores nulos en la base de datos.

3.19. Algunos sistemas permiten los valores nulos *marcados*. Un valor nulo marcado $\perp_i$ es igual a sí mismo, pero si $i \neq j$, $\perp_i \neq \perp_j$. Una aplicación de valores nulos marcados debe permitir ciertas actualizaciones mediante el uso de vistas. Considérese la vista *información-crédito* (Apartado 3.5). Muéstrese la manera en que se pueden utilizar los valores nulos marcados para permitir la inserción de la tupla («González», 1900) mediante *información-crédito*.

NOTAS BIBLIOGRÁFICAS

El modelo relacional fue propuesto por E. F. Codd del Laboratorio de investigación de San José de IBM a finales de los años sesenta [Codd, 1970]. Este trabajo motivó la concesión a Codd del prestigioso Premio Turing de la ACM en 1981 (Codd [1982]).

Siguiendo el trabajo original de Codd se constituyeron varios proyectos de investigación con el objetivo de crear sistemas de bases de datos relacionales prácticos, incluyendo System R del Laboratorio de investigación de San José de IBM, Ingres en la Universidad de California en Berkeley, Query-by-Example en el Centro de investigación T. J. Watson de IBM (*IBM T. J. Watson Research Center*) y el vehículo de prueba relacional (*Peterlee Relational Test Vehicle*, PRTV) del Centro científico de IBM (*IBM Scientific Center*) en Peterlee, Reino Unido. System R se discute en Astrahan et al. [1976], Astrahan et al. [1979] y en Chamberlin et al. [1981]. Ingres se discute en Stonebraker [1980], Stonebraker [1986b] y en Stonebraker et al. [1976]. Query-by-Example se describe en Zloof [1977]. PRTV se describe en Todd [1976].

Actualmente están disponibles comercialmente numerosos productos de bases de datos relacionales. Ejemplos de ello son DB2 de IBM, Ingres, Oracle, Sybase, Informix y Microsoft SQL Server. Ejemplos de productos de bases de datos para las computadoras personales son Microsoft Access, dBase y FoxPro. La información sobre estos productos puede hallarse en sus manuales respectivos.

En la mayor parte de los textos sobre bases de datos se incluye una discusión general del modelo relacional de datos. Atzeni y De Antonellis [1993] y Maier [1983] son textos dedicados exclusivamente al modelo relacional de datos. La definición original del álgebra relacional está en Codd [1970]; la del cálculo relacional de tuplas en Codd [1972]. En Codd [1972] se encuentra una prueba formal de la equivalencia del cálculo relacional de tuplas y del álgebra relacional.

Se han propuesto varias ampliaciones del cálculo relacional. Klug [1982] y Escobar-Molano et al. [1993] describen ampliaciones para funciones de agregación escalares. En Codd [1979] se presentan ampliaciones del modelo relacional y discusiones sobre la incorporación de los valores nulos al álgebra relacional (el modelo RM/T), así como las la de las reuniones externas. Codd [1990] es un compendio de los trabajos de E. F. Codd sobre el modelo relacional. Las reuniones externas también se discuten en Date [1993b]. El problema de la actualización de las bases de datos relacionales mediante vistas se aborda en Bancilhon y Spyratos [1981], Cosmadakis y Papadimitriou [1984], Dayal y Bernstein [1978, 1982] y Langerak [1990]. El Apartado 14.5 trata el mantenimiento de las vistas materializadas, y las referencias a la literatura sobre ello se pueden encontrar al final de ese capítulo.



BASES DE DATOS RELACIONALES

Una base de datos relacional es un repositorio compartido de datos. Para hacer disponibles los datos de un base de datos relacional a los usuarios hay que considerar varios aspectos. Uno es la forma en que los usuarios solicitan los datos: ¿cuáles son los diferentes lenguajes de consulta que usan? El Capítulo 4 trata el lenguaje SQL, que es el lenguaje de consulta más ampliamente usado actualmente. El Capítulo 5 trata otros dos lenguajes de consulta, QBE y Datalog, que ofrecen enfoques alternativos a la consulta de datos relacionales.

Otro aspecto es la integridad de datos y la seguridad; las bases de datos necesitan proteger los datos del daño provocado por los usuarios, ya sean intencionados o no. El componente de mantenimiento de la integridad de una base de datos asegura que las actualizaciones no violan las restricciones de integridad que hayan especificado sobre los datos. El componente de seguridad de una base de datos incluye la autenticación de usuarios y el control de acceso para restringir las posibles acciones de cada usuario. El Capítulo 6 trata los aspectos de integridad y seguridad. Estos aspectos se presentan independientemente del modelo de datos, pero se estudian en el contexto del modelo de datos relacional para ejemplificarlos. Las restricciones de integridad forman la base del diseño de bases de datos relacionales, que se estudian en el Capítulo 7.

El diseño de bases de datos relacionales —el diseño del esquema relacional— es el primer paso en la construcción de aplicaciones de bases de datos. El diseño de esquemas se trató informalmente en capítulos anteriores. Sin embargo, hay principios que se pueden usar para distinguir los buenos diseños de bases de datos. Se formalizan mediante varias «formas normales», que ofrecen diferentes compromisos entre la posibilidad de inconsistencias y la eficiencia de ciertas consultas. El Capítulo 7 describe el diseño formal de esquemas relacionales.

CAPÍTULO

4 SQL

Los lenguajes formales descritos en el Capítulo 3 proporcionan una notación concisa para la representación de consultas. Sin embargo, los sistemas de bases de datos comerciales necesitan un lenguaje de consultas cómodo para el usuario. En este capítulo se estudia el lenguaje comercial de mayor influencia, SQL. SQL usa una combinación de álgebra relacional y construcciones del cálculo relacional.

Aunque el lenguaje SQL se considere un lenguaje de consultas, contiene muchas otras capacidades además de la consulta en bases de datos. Incluye características para definir la estructura de los datos, para la modificación de los datos en la base de datos y para la especificación de restricciones de seguridad.

No se pretende proporcionar un manual de usuario completo para SQL. Por el contrario, se presentan las construcciones y conceptos fundamentales de SQL. Las distintas implementaciones de SQL pueden diferenciarse en detalles, o pueden admitir sólo un subconjunto del lenguaje completo.

4.1. INTRODUCCIÓN

IBM desarrolló la versión original en su Laboratorio de Investigación de San José (*San José Research Center*, actualmente Centro de Investigación de Almadén, *Almadén Research Center*). IBM implementó el lenguaje, originalmente denominado Sequel, como parte del proyecto System R, a principios de 1970. El lenguaje Sequel ha evolucionado desde entonces y su nombre ha pasado a ser SQL (*Structured Query Language*, Lenguaje estructurado de consultas). Actualmente, numerosos productos son compatibles con el lenguaje SQL. SQL se ha establecido como *el* lenguaje estándar de bases de datos relacionales.

En 1986, ANSI (*American National Standards Institute*, Instituto Nacional Americano de Normalización) e ISO (*International Standards Organization*, Organización Internacional de Normalización), publicaron una norma SQL, denominada SQL-86. En 1987, IBM publicó su propia norma de SQL corporativo, Interfaz de bases de datos para arquitecturas de aplicación a sistemas (*Systems Application Architecture Database Interface*, SAA-SQL). En 1989 se publicó una norma extendida para SQL denominada SQL-89 y actualmente los sistemas de bases de datos son normalmente compatibles al menos con las características de SQL-89. La siguiente versión de la norma fue SQL-92 y la versión más reciente es SQL:1999. Las notas bibliográficas proporcionan referencias a esas normas.

En este apartado se presenta una visión general de SQL basada en la norma SQL-92 ampliamente implementada. La norma SQL:1999 es un superconjunto de la norma SQL-92; en este capítulo se tratan algunas características de SQL:1999 y se proporciona un estudio más detallado en el Capítulo 9. Muchos sistemas de bases de datos soportan algunas de las nuevas constructoras de SQL:1999, aunque ningún sistema de bases datos actual soporta todas las nuevas constructoras. También hay ser consciente de que algunos sistemas de bases de datos ni siquiera soportan todas las características de SQL-92 y de que muchas bases de datos proporcionan características no estándar que no se tratan aquí.

El lenguaje SQL tiene varios componentes:

- **Lenguaje de definición de datos** (LDD). El LDD de SQL proporciona órdenes para la definición de esquemas de relación, borrado de relaciones, creación de índices y modificación de esquemas de relación.
- **Lenguaje interactivo de manipulación de datos** (LMD). El LMD de SQL incluye un lenguaje de consultas, basado tanto en el álgebra relacional como en el cálculo relacional de tuplas. Incluye también órdenes para insertar, borrar y modificar tuplas de la base de datos.
- **Definición de vistas.** El LDD de SQL incluye órdenes para la definición de vistas.
- **Control de transacciones.** SQL incluye órdenes para la especificación del comienzo y final de transacciones.
- **SQL incorporado** y **SQL dinámico.** SQL dinámico e incorporado define cómo se pueden incorporar las instrucciones SQL en lenguajes de pro-

gramación de propósito general, tales como C, C++, Java, PL/I, Cobol, Pascal y Fortran.

- **Integridad.** El LDD de SQL incluye órdenes para la especificación de las restricciones de integridad que deben satisfacer los datos almacenados en la base de datos. Las actualizaciones que violen las restricciones de integridad se rechazan.
- **Autorización.** El LDD de SQL incluye órdenes para especificar derechos de acceso para las relaciones y vistas.

En este capítulo se estudia el LMD y las características básicas del LDD de SQL. También se describe brevemente SQL incorporado y dinámico, incluyendo las normas ODBC y JDBC para la interacción con una base de datos desde programas escritos en lenguajes C y Java. Las características de SQL que dan soporte a la integridad y autorización se describen en el Capítulo 6, mientras que el Capítulo 9 esboza las extensiones orientadas a objeto de SQL.

Los ejemplos de este capítulo y posteriores se basarán en una empresa bancaria, con los siguientes esquemas de relación:

Esquema-sucursal = (*nombre-sucursal*, *ciudad-sucursal*, *activo*)
Esquema-cliente = (*nombre-cliente*, *calle-cliente*, *ciudad-cliente*)
Esquema-préstamo = (*número-préstamo*, *nombre-sucursal*, *importe*)
Esquema-prestatario = (*nombre-cliente*, *número-préstamo*)
Esquema-cuenta = (*número-cuenta*, *nombre-sucursal*, *saldo*)
Esquema-impositor = (*nombre-cliente*, *número-cuenta*)

Nótese que en este capítulo, como en el resto del texto, se usan nombres separados por guiones para los esquemas, relaciones y atributos para facilitar su lectura. Sin embargo, en los sistemas SQL actuales, los guiones no son partes válidas de un nombre (se tratan como el operador menos). Una forma simple de traducir los nombres que se usan aquí a nombres SQL válidos es reemplazar todos los guiones por el símbolo de subrayado («_»). Por ejemplo, se usa *nombre_sucursal* en lugar de *nombre-sucursal*.

4.2. ESTRUCTURA BÁSICA

Una base de datos relacional consiste en un conjunto de relaciones, a cada una de las cuales se le asigna un nombre único. Cada relación tiene una estructura similar a la presentada en el Capítulo 3. SQL permite el uso de valores nulos para indicar que el valor o bien es desconocido, o no existe. Se fijan criterios que permiten al usuario especificar a qué atributos no se puede asignar valor nulo, como estudiaremos en el Apartado 4.11.

La estructura básica de una expresión SQL consiste en tres cláusulas: **select**, **from** y **where**.

- La cláusula **select** corresponde a la operación proyección del álgebra relacional. Se usa para listar los atributos deseados del resultado de una consulta.
- La cláusula **from** corresponde a la operación producto cartesiano del álgebra relacional. Lista las relaciones que deben ser analizadas en la evaluación de la expresión.
- La cláusula **where** corresponde al predicado selección del álgebra relacional. Es un predicado que engloba a los atributos de las relaciones que aparecen en la cláusula **from**.

Un hecho histórico desafortunado es que el término *select* tiene un significado diferente en SQL que en el álgebra relacional. A continuación se resaltan las diferentes interpretaciones, a fin de minimizar la posible confusión.

Una consulta típica en SQL tiene la forma

select $A_1, A_2, \ldots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

Cada *Ai* representa un atributo, y cada *ri* una relación. *P* es un predicado. La consulta es equivalente a la expresión del álgebra relacional

$$\Pi\ A_1, A_2, \ldots, A_n\ (\sigma_P\ (r_1 \times r_2 \times \ldots \times r_m))$$

Si se omite la cláusula **where**, el predicado P es **cierto**. Sin embargo, con diferencia a la expresión del álgebra relacional, el resultado de la consulta SQL puede contener varias copias de algunas tuplas; este aspecto se analizará de nuevo en el Apartado 4.2.8.

SQL forma el producto cartesiano de las relaciones incluidas en la cláusula **from**, lleva a cabo la selección del álgebra relacional usando el predicado de la cláusula **where** y entonces proyecta el resultado sobre los atributos de la cláusula **select**. En la práctica, SQL puede convertir la expresión en una forma equivalente que puede ser procesada más eficientemente. Las cuestiones relativas a la eficiencia se analizan en los Capítulos 13 y 14.

4.2.1. La cláusula select

El resultado de una consulta SQL es, por supuesto, una relación. Considérese una consulta simple, usando el

ejemplo bancario, «Obtener los números de todas las sucursales en la relación *préstamo*»:

```
select nombre-sucursal
from préstamo
```

El resultado es una relación consistente en el único atributo *nombre-sucursal*.

Los lenguajes formales de consulta están basados en la noción matemática de que una relación es un conjunto. Así, nunca aparecen tuplas duplicadas en las relaciones. En la práctica, la eliminación de duplicados consume tiempo. Sin embargo, SQL (como la mayoría de los lenguajes de consulta comerciales) permite duplicados en las relaciones, así como en el resultado de las expresiones SQL. Así, la consulta anterior listará cada *nombre-sucursal* una vez por cada tupla en la que aparece en la relación *préstamo*.

En aquellos casos donde se quiera forzar la eliminación de duplicados, se insertará la palabra clave **distinct** después de **select**. Por lo tanto, se puede reescribir la consulta anterior como

```
select distinct nombre-sucursal
from préstamo
```

si se desean eliminar los duplicados.

Es importante resaltar que SQL permite usar la palabra clave **all** para especificar explícitamente que no se eliminan los duplicados:

```
select all nombre-sucursal
from préstamo
```

Como de manera predeterminada se realiza la retención de duplicados, de ahora en adelante no se usará la palabra clave **all** en los ejemplos. Para asegurar la eliminación de duplicados en el resultado de los ejemplos de consultas, se usará la cláusula **distinct** siempre que sea necesario. En la mayoría de las consultas donde no se utiliza **distinct**, el número exacto de copias duplicadas de cada tupla que resultan de la consulta no es importante. Sin embargo, el número es importante en ciertas aplicaciones; este aspecto se volverá a tratar en el Apartado 4.2.8.

El símbolo asterisco «*» se puede usar para denotar «todos los atributos». Así, el uso de *préstamo*.* en la cláusula **select** anterior indicaría que todos los atributos de préstamo serían seleccionados. Una cláusula **select** de la forma **select** * indica que se deben seleccionar todos los atributos de todas las relaciones que aparecen en la cláusula **from**.

La cláusula **select** puede contener también expresiones aritméticas que contengan los operadores, +, –, ∗ y / operando sobre constantes o atributos de la tuplas. Por ejemplo, la consulta

```
select nombre-sucursal, número-préstamo,
       importe * 100
from préstamo
```

devolverá una relación que es igual que la relación *préstamo*, salvo que el atributo *importe* está multiplicado por 100.

SQL también proporciona tipos de datos especiales, tales como varias formas del tipo *fecha* y permite varias funciones aritméticas para operar sobre esos tipos.

4.2.2. La cláusula where

A continuación se ilustra con un ejemplo el uso de la cláusula **where** en SQL. Considérese la consulta «Obtener todos los números de préstamo para préstamos hechos en la sucursal con nombre Navacerrada, en los que el importe sea superior a 1.200 €». Esta consulta puede escribirse en SQL como

```
select número-préstamo
from préstamo
where nombre-sucursal = 'Navacerrada' and
      importe > 1200
```

SQL usa las conectivas lógicas **and**, **or** y **not** (en lugar de los símbolos matemáticos ∧, ∨ y ¬) en la cláusula **where**. Los operandos de las conectivas lógicas pueden ser expresiones que contengan los operadores de comparación <, <=, >, >=, = y <>. SQL permite usar los operadores de comparación para comparar cadenas y expresiones aritméticas, así como tipos especiales, tales como el tipo fecha.

SQL incluye un operador de comparación **between** para simplificar las cláusulas **where** que especifica que un valor sea menor o igual que un valor y mayor o igual que otro valor. Si se desea obtener el número de préstamo de aquellos préstamos por importes entre 90.000 € y 100.000 €, se puede usar la comparación **between** para escribir

```
select número-préstamo
from préstamo
where importe between 90000 and 100000
```

en lugar de

```
select número-préstamo
from préstamo
where importe <= 100000 and importe >= 90000
```

De forma análoga, se puede usar el operador de comparación **not between**.

4.2.3. La cláusula from

Finalmente, se estudia el uso de la cláusula **from**. La cláusula **from** define por sí misma un producto cartesiano de las relaciones que aparecen en la cláusula. Escribir una expresión SQL para la reunión natural es una tarea relativamente fácil, puesto que la reunión natu-

ral se define en términos de un producto cartesiano, una selección y una proyección.

La expresión del álgebra relacional se escribe como sigue:

$$\Pi_{nombre\text{-}cliente,\ número\text{-}préstamo, importe}(prestatario \bowtie préstamo)$$

para la consulta «Para todos los clientes que tienen un préstamo en el banco, obtener los nombres, números de préstamo e importes». Esta consulta puede escribirse en SQL como

select *nombre-cliente*, *prestatario.número-préstamo*,
importe
from *prestatario*, *préstamo*
where *prestatario.número-préstamo*
= *préstamo.número-préstamo*

Nótese que SQL usa la notación *nombre-relación.nombre-atributo*, como lo hace el álgebra relacional, para evitar ambigüedad en los casos en que un atributo aparece en el esquema de más de una relación. También se podría haber escrito *prestatario.nombre-cliente* en lugar de *nombre-cliente*, en la cláusula **select**. Sin embargo, como el atributo *nombre-cliente* aparece sólo en una de las relaciones de la cláusula **from**, no existe ambigüedad al escribir *nombre-cliente*.

Se puede extender la consulta anterior y considerar un caso más complicado en el que se pide además qué clientes poseen un préstamo en la sucursal Navacerrada: «Obtener los nombres, números de préstamo e importes de todos los clientes que tienen un préstamo en la sucursal Navacerrada». Para escribir esta consulta será necesario establecer dos restricciones en la cláusula **where**, relacionadas con la conectiva lógica **and**:

select *nombre-cliente*, *prestatario.número-préstamo*,
importe
from *prestatario*, *préstamo*
where *prestatario.número-préstamo* =
préstamo.número-préstamo **and**
nombre-sucursal= 'Navacerrada'

SQL-92 incluye extensiones para llevar a cabo reuniones naturales y reuniones externas en la cláusula **from**. Esto se estudiará en el Apartado 4.10.

4.2.4. La operación renombramiento

SQL proporciona un mecanismo para renombrar tanto relaciones como atributos. Para ello utiliza la cláusula **as**, que tiene la forma siguiente:

nombre-antiguo **as** *nombre-nuevo*

la cláusula **as** puede aparecer tanto en **select** como en **from**.

Considérese de nuevo la consulta anterior:

select distinct *nombre-cliente*, *prestatario.número-préstamo*, *importe*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo*
= *préstamo.número-préstamo*

El resultado de esta consulta es una relación con los atributos siguientes:

nombre-cliente, *número-préstamo*, *importe*.

Los nombres de los atributos en el resultado se derivan de los nombres de los atributos de la relación que aparece en la cláusula **from**.

Sin embargo, no se pueden derivar siempre los nombres de este modo. En primer lugar, dos relaciones que aparecen en la cláusula **from** pueden tener atributos con el mismo nombre, en cuyo caso, un nombre de atributo se duplica en el resultado. En segundo lugar, si se incluye una expresión aritmética en la cláusula **select**, los atributos resultantes no tienen el mismo nombre. Y en tercer lugar, incluso si un nombre de atributo se puede derivar de las relaciones base, como en el ejemplo anterior, se puede querer cambiar el nombre del atributo en el resultado. Para todo ello, SQL proporciona una forma de renombrar los atributos de una relación resultado.

Por ejemplo, si se quisiera renombrar el atributo *número-préstamo*, asociándole el nombre de *id-préstamo*, se podría reescribir la consulta anterior del siguiente modo

select *nombre-cliente*, *prestatario.número-préstamo*
as *id-préstamo, importe*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo* =
préstamo.número-préstamo

4.2.5. Variables tupla

La cláusula **as** es particularmente útil en la definición del concepto de variables tupla, como se hace en el cálculo relacional de tuplas. Una variable tupla en SQL se debe asociar con una relación concreta. Las variables tupla se definen en la cláusula **from** mediante el uso de la cláusula **as**. Como ejemplo, a continuación se reescribe la consulta «Obtener los nombres y números de préstamo de todos los clientes que tienen un préstamo en el banco» como sigue

select *nombre-cliente*, *T.número-préstamo*, *S.importe*
from *prestatario* **as** *T*, *préstamo* **as** *S*
where *T.número-préstamo* = *S.número-préstamo*

Nótese que se define la variable tupla en la cláusula **from**, colocándola después del nombre de la relación a la cual está asociada y detrás de la palabra clave **as** (la palabra clave **as** es opcional). Al escribir expresiones

de la forma *nombre-relación.nombre-atributo*, el nombre de la relación es, en efecto, una variable tupla definida implícitamente.

Las variables tupla son de gran utilidad para comparar dos tuplas de la misma relación. Hay que recordar que, en los casos de este tipo, se puede usar la operación renombramiento del álgebra relacional. Si se desea formular la consulta «Obtener los nombres de todas las sucursales que poseen un activo mayor que al menos una sucursal situada en Barcelona», se puede escribir la siguiente expresión SQL

select distinct *T.nombre-sucursal*
from *sucursal* **as** *T*, *sucursal* **as** *S*
where *T.activo* > *S.activo* **and** *S.ciudad-sucursal* = 'Barcelona'

Obsérvese que no se puede utilizar la notación *sucursal.activo*, puesto que no estaría claro a qué aparición de *sucursal* se refiere.

SQL permite usar la notación $(v_1, v_2, \ldots, v_n)$ para designar una tupla de aridad n que contiene los valores $v_1, v_2, \ldots, v_n$. Los operadores de comparación se pueden utilizar sobre tuplas, y el orden se define lexicográficamente. Por ejemplo $(a_1, a_2) \leq (b_1, b_2)$ es cierto si $(a_1 < b_1)$ o si se cumple que $(a_1 = b_1) \wedge (a_2 \leq b_2)$; análogamente, dos tuplas son iguales si lo son todos sus atributos.

4.2.6. Operaciones sobre cadenas

SQL especifica las cadenas encerrándolas entre comillas simple, como 'Navacerrada', como se vio anteriormente. Un carácter comilla que sea parte de una cadena se puede especificar usando dos caracteres comilla; por ejemplo, la cadena «El carácter ' se puede ver en esta cadena» se puede especificar como 'El carácter '' se puede ver en esta cadena'.

La operación más usada sobre cadenas es el encaje de patrones, para el que se usa el operador **like**. Para la descripción de patrones se utilizan los dos caracteres especiales siguientes:

- Tanto por ciento (%): El carácter % encaja con cualquier subcadena.
- Subrayado (_): El carácter _ encaja con cualquier carácter.

Los patrones son muy sensibles, esto es, los caracteres en mayúsculas no encajan con los caracteres en minúscula, o viceversa. Para ilustrar el encaje de patrones, considérense los siguientes ejemplos:

- 'Nava%' encaja con cualquier cadena que empiece con «Nava».
- '%cer%' encaja con cualquier cadena que contenga «cer» como subcadena, por ejemplo 'Navacerrada', 'Cáceres' y 'Becerril'.
- '_ _ _' encaja con cualquier cadena de tres caracteres.
- '_ _ _%' encaja con cualquier cadena de al menos tres caracteres.

Los patrones se expresan en SQL utilizando el operador de comparación **like**. Considérese la consulta siguiente: «Obtener los nombres de todos los clientes cuyas calles contengan la subcadena 'Mayor'». Esta consulta se podría escribir como sigue

select *nombre-cliente*
from *cliente*
where *calle-cliente* **like** '%Mayor%'

Para que los patrones puedan contener los caracteres especiales patrón (esto es, % y _), SQL permite la especificación de un carácter de escape. El carácter de escape se utiliza inmediatamente antes de un carácter especial patrón para indicar que ese carácter especial va a ser tratado como un carácter normal. El carácter de escape para una comparación **like** se define utilizando la palabra clave **escape**. Para ilustrar esto, considérense los siguientes patrones, los cuales utilizan una barra invertida (\) como carácter de escape:

- **like** 'ab\%cd%' **escape** '\' encaja con todas las cadenas que empiecen por ab%cd .
- **like** 'ab\\cd%' **escape** '\' encaja con todas las cadenas que empiecen por ab\cd .

SQL permite buscar discordancias en lugar de concordancias utilizando el operador de comparación **not like**.

SQL también proporciona una variedad de funciones que operan sobre cadenas de caracteres, tales como la concatenación (usando «||»), la extracción de subcadenas, el cálculo de la longitud de las cadenas, la conversión a mayúsculas y minúsculas, etc. SQL:1999 también ofrece una operación **similar to** que proporciona un encaje de patrones más potente que la operación **like**; la sintaxis para especificar patrones es similar a la usada en Unix para expresiones regulares.

4.2.7. Orden en la presentación de las tuplas

SQL ofrece al usuario cierto control sobre el orden en el cual se presentan las tuplas de una relación. La cláusula **order by** hace que las tuplas resultantes de una consulta se presenten en un cierto orden. Para listar en orden alfabético todos los clientes que tienen un préstamo en la sucursal Navacerrada se escribirá:

select distinct *nombre_cliente*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo* = *préstamo.número-préstamo* **and** *nombre-sucursal* = 'Navacerrada'
order by *nombre-cliente*

De manera predeterminada la cláusula **order by** lista los elementos en orden ascendente. Para especificar el tipo de ordenación se puede incluir la cláusula **desc** para orden descendente o **asc** para orden ascendente. Además, se puede ordenar con respecto a más de un atributo. Si se desea listar la relación *préstamo* en orden descendente para *importe*. Si varios préstamos tienen el mismo importe, se ordenan ascendentemente según el número de préstamo. Esta consulta en SQL se escribe del modo siguiente:

select *
from *préstamo*
order by *importe* **desc**, *número-préstamo* **asc**

Para ejecutar una consulta que contiene la cláusula **order by**, SQL tiene que llevar a cabo una ordenación. Como ordenar un gran número de tuplas puede ser costoso, es conveniente ordenar sólo cuando sea estrictamente necesario.

4.2.8. Duplicados

La utilización de relaciones con duplicados se ha mostrado útil en diversas situaciones. SQL no sólo define formalmente las tuplas que están en el resultado de una consulta, sino también el número de copias de cada una de esas tuplas que aparece en el resultado. La semántica de duplicados de una consulta SQL se puede definir utilizando versiones de los operadores relacionales para *multiconjuntos*. A continuación se definen las versiones multiconjunto de varios de los operadores del álgebra relacional. Dadas las relaciones multiconjunto r_1 y r_2,

1. Si existen c_1 copias de la tupla t_1 en r_1, y t_1 satisface la selección σ_θ, entonces hay c_1 copias de t_1 en $\sigma_\theta(r_1)$.
2. Para cada copia de la tupla t_1 en r_1, hay una copia de la tupla $\Pi_A(t_1)$ en $\Pi_A(r_1)$, donde $\Pi_A(t_1)$ denota la proyección de la tupla única t_1.
3. Si existen c_1 copias de la tupla t_1 en r_1 y c_2 copias de la tupla t_2 en r_2, entonces hay $c_1 * c_2$ copias de la tupla $t_1.t_2$ en $r_1 \times r_2$.

Por ejemplo, supóngase que las relaciones r_1 con esquema (A, B) y r_2 con esquema (C) son los multiconjuntos siguientes:

$$r_1 = \{(1,a), (2,a)\} \qquad r_2 = \{(2), (3), (3)\}$$

Entonces, $\Pi_B(r_1)$ sería $\{(a), (a)\}$, mientras que $\Pi_B(r_1) \times r_2$ sería

$$\{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\}$$

Se puede ahora definir cuántas copias de cada tupla aparecen en el resultado de una consulta SQL. Una consulta SQL de la forma

select $A_1, A_2, \ldots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

es equivalente a la expresión del álgebra relacional

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

usando las versiones multiconjunto de los operadores relacionales σ, Π y $\times$.

4.3. OPERACIONES SOBRE CONJUNTOS

Las operaciones de SQL-92 **union**, **intersect** y **except** operan sobre relaciones y corresponden a las operaciones del álgebra relacional ∪, ∩ y –. Al igual que la unión, intersección y diferencia de conjuntos en el álgebra relacional, las relaciones que participan en las operaciones han de ser *compatibles*; esto es, deben tener el mismo conjunto de atributos.

A continuación se demuestra cómo se pueden formular en SQL varias de las consultas de ejemplo consideradas en el Capítulo 3 utilizando consultas que incluyen las operaciones **union**, **intersect** y **except** de dos conjuntos. Los dos conjuntos utilizados serán: el conjuntos de todos los clientes que tienen una cuenta en el banco, que puede obtenerse con:

select *nombre-cliente*
from *impositor*

y el conjunto de todos los clientes que tienen un préstamo en el banco, que puede obtenerse con:

select *nombre-cliente*
from *prestatario*

A partir de ahora, las letras *i* y *p* se utilizarán para hacer referencia a las relaciones obtenidas como resultado de las dos consultas anteriores.

4.3.1. La operacion unión

Para encontrar todos los clientes que poseen un préstamo, una cuenta o las dos cosas en el banco, se escribirá:

(**select** *nombre-cliente*
from *impositor*)

```
union
(select nombre-cliente
 from prestatario)
```

A diferencia de la cláusula **select**, la operación **union** (unión) elimina duplicados automáticamente. Así, en la consulta anterior, si un cliente —por ejemplo, Santos— tiene varias cuentas o préstamos (o ambas cosas) en el banco, entonces Santos aparecerá sólo una vez en el resultado.

Para conservar los duplicados, se utilizará **union all** en lugar de **union**:

```
(select nombre-cliente
 from impositor)
union all
(select nombre-cliente
 from prestatario)
```

El número de tuplas duplicadas en el resultado es igual al número total de duplicados que aparecen en *i* y *p*. Así, si Santos tuviese tres cuentas y dos préstamos en el banco, entonces en el resultado aparecerían cinco tuplas con el nombre de Santos.

4.3.2. La operación intersección

Para encontrar todos los clientes que tienen tanto un préstamo como una cuenta en el banco, se escribirá:

```
(select distinct nombre-cliente
 from impositor)
intersect
(select distinct nombre-cliente
 from prestatario)
```

La operacion **intersect** (intersección) elimina duplicados automáticamente. Así, en la consulta anterior, si un cliente —por ejemplo, Santos— tiene varias cuentas o préstamos (o ambas cosas) en el banco, entonces Santos aparecerá solo una vez en el resultado.

Para conservar los duplicados se utilizará **intersect all** en lugar de **intersect**:

```
(select nombre-cliente
 from impositor)
intersect all
(select nombre-cliente
 from prestatario)
```

El número de tuplas duplicadas en el resultado es igual al mínimo número de duplicados que aparecen en *i* y *p*. Así, si Santos tuviese tres cuentas y dos préstamos en el banco, entonces en el resultado de la consulta aparecerían dos tuplas con el nombre de Santos.

4.3.3. La operación excepto

Para encontrar todos los clientes que tienen cuenta pero no tienen ningún préstamo en el banco se escribirá:

```
(select distinct nombre-cliente
 from impositor)
except
(select distinct nombre-cliente
 from prestatario)
```

La operacion **except** (excepto) elimina duplicados automáticamente. Así, en la consulta anterior, una tupla con el nombre de Santos aparecerá en el resultado (exactamente una vez), sólo si Santos tiene una cuenta en el banco, pero no tiene ningún préstamo en el mismo.

Para conservar los duplicados, se utilizará **except all** en lugar de **except**:

```
(select nombre-cliente
 from impositor)
except all
(select nombre-cliente
 from prestatario)
```

El número de copias duplicadas de una tupla en el resultado es igual al número de copias duplicadas de dicha tupla en *i* menos el número de copias duplicadas de la misma tupla en *p*, siempre que la diferencia sea positiva. Así, si Santos tuviese tres cuentas y un préstamo en el banco, entonces en el resultado aparecerían dos tuplas con el nombre de Santos. Si, por el contrario, dicho cliente tuviese dos cuentas y tres préstamos en el banco, no habrá ninguna tupla con el nombre de Santos en el resultado.

4.4. FUNCIONES DE AGREGACIÓN

Las funciones de agregación son funciones que toman una colección (un conjunto o multiconjunto) de valores como entrada y producen un único valor como salida. SQL proporciona cinco funciones de agregación primitivas:

- Media: **avg**
- Mínimo: **min**
- Máximo: **max**
- Total: **sum**
- Cuenta: **count**

La entrada a **sum** y **avg** debe ser una colección de números, pero los otros operadores pueden operar sobre colecciones de datos de tipo no numérico, tales como las cadenas.

Como ejemplo, considérese la consulta «Obtener la media de saldos de las cuentas de la sucursal Navacerrada». Esta consulta se puede formular del modo siguiente:

select avg (*saldo*)
from *cuenta*
where *nombre-sucursal* = 'Navacerrada'

El resultado de esta consulta será una relación con un único atributo, que contendrá una única fila con un valor numérico correspondiente al saldo medio de la sucursal Navacerrada. Opcionalmente se puede dar un nombre al atributo resultado de la relación, usando la cláusula **as**.

Existen situaciones en las cuales sería deseable aplicar las funciones de agregación no sólo a un único conjunto de tuplas sino también a un grupo de conjuntos de tuplas; esto se especifica en SQL usando la cláusula **group by**. El atributo o atributos especificados en la cláusula **group by** se usan para formar grupos. Las tuplas con el mismo valor en todos los atributos especificados en la cláusula **group by** se colocan en un grupo.

Como ejemplo, considérese la consulta «Obtener el saldo medio de las cuentas de cada sucursal».

Dicha consulta se formulará del modo siguiente

select *nombre-sucursal*, **avg** (*saldo*)
from *cuenta*
group by *nombre-sucursal*

La conservación de duplicados es importante al calcular una media. Supóngase que los saldos de las cuentas en la (pequeña) sucursal de nombre «Galapagar» son 1.000 €, 3.000 €, 2.000 € y 1.000 €. El saldo medio es 7.000/4 =1.750 €. Si se eliminasen duplicados se obtendría un resultado erróneo (6.000/3 = 2.000 €).

Hay casos en los que se deben eliminar los duplicados antes de calcular una función de agregación. Para eliminar duplicados se utiliza la palabra clave **distinct** en la expresión de agregación. Como ejemplo considérese la consulta «Obtener el número de impositores de cada sucursal». En este caso un impositor sólo se debe contar una vez, sin tener en cuenta el número de cuentas que el impositor pueda tener. La consulta se formulará del modo siguiente:

select *nombre-sucursal*, **count** (**distinct** *nombre-cliente*)
from *impositor*, *cuenta*
where *impositor.número-cuenta* = *cuenta.número-cuenta*
group by *nombre-sucursal*

A veces es más útil establecer una condición que se aplique a los grupos que una que se aplique a las tuplas. Por ejemplo, podemos estar interesados sólo en aquellas sucursales donde el saldo medio de cuentas es superior a 1.200 €. Esta condición no es aplicable a una única tupla; se aplica a cada grupo construido por la cláusula **group by**. Para expresar este tipo de consultas se utiliza la cláusula **having** de SQL. Los predicados de la cláusula **having** se aplican después de la formación de grupos, de modo que se pueden usar las funciones de agregación. Esta consulta se expresa en SQL del modo siguiente:

select *nombre-sucursal*, **avg** (*saldo*)
from *cuenta*
group by *nombre-sucursal*
having avg (saldo) > 1200

A veces se desea tratar la relación entera como un único grupo. En casos de este tipo no se usa la cláusula **group by**. Considérese la consulta «Obtener el saldo medio de todas las cuentas». Esta consulta se formulará del modo siguiente:

select avg (*saldo*)
from *cuenta*

Con mucha frecuencia se usa la función de agregación **count** para contar el número de tuplas de una relación. La notación para esta función en SQL es **count** (*). Así, para encontrar el número de tuplas de la relación *cliente*, se escribirá

select count (*)
from *cliente*

SQL no permite el uso de **distinct** con **count** (*). Sí se permite, sin embargo, el uso de **distinct** con **max** y **min**, incluso cuando el resultado no cambia. Se puede usar la palabra clave **all** en lugar de **distinct** para especificar la retención de duplicados, pero como **all** se especifica de manera predeterminada, no es necesario incluir dicha cláusula.

Si en una misma consulta aparece una cláusula **where** y una cláusula **having**, se aplica primero el predicado de la cláusula **where**. Las tuplas que satisfagan el predicado de la cláusula **where** se colocan en grupos según la cláusula **group by**. La cláusula **having**, si existe, se aplica entonces a cada grupo; los grupos que no satisfagan el predicado de la cláusula **having** se eliminan. La cláusula **select** utiliza los grupos restantes para generar las tuplas resultado de la consulta.

Para ilustrar el uso de la cláusula **where** y la cláusula **having** dentro de la misma consulta considérese el ejemplo «Obtener el saldo medio de cada cliente que vive en Madrid y tiene como mínimo tres cuentas».

select *impositor.nombre-cliente*, **avg** (*saldo*)
from *impositor*, *cuenta*, *cliente*
where *impositor.número-cuenta*
= *cuenta.número-cuenta* **and**
impositor.nombre-cliente
= *cliente.nombre-cliente* **and**
ciudad-cliente = 'Madrid'
group by *impositor.nombre-cliente*
having count (**distinct** *impositor.número-cuenta*) >= 3

4.5. VALORES NULOS

SQL permite el uso de valores nulos para indicar la ausencia de información sobre el valor de un atributo.

En un predicado se puede usar la palabra clave especial **null** para comprobar si un valor es nulo. Así, para encontrar todos los números de préstamo que aparecen en la relación *préstamo* con valores nulos para *importe* se escribe

select *número-préstamo*
from *préstamo*
where *importe* **is null**

El predicado **is not null** pregunta por la ausencia de un valor nulo.

El uso de un valor nulo en las operaciones aritméticas y de comparación causa varias complicaciones. En el Apartado 3.3.4 se vio cómo se manejan los valores nulos en el álgebra relacional. Ahora se describe cómo maneja SQL los valores nulos.

El resultado de una expresión aritmética (incluyendo por ejemplo +,–,* o /) es nulo si cualquiera de los valores de entrada es nulo. SQL trata como **desconocido** el resultado de cualquier comparación que implique un valor *nulo* (aparte de **is null** e **is not null**).

Dado que el predicado en una cláusula **where** puede incluir operaciones booleanas tales como **and**, **or** y **not** sobre los resultados de las comparaciones, las definiciones de estas operaciones se extienden para manejar el valor **desconocido**, como se describe en el Apartado 3.3.4.

- **and**: el resultado de *cierto* **and** *desconocido* es *desconocido*, *falso* **and** *desconocido* es *falso*, mientras que *desconocido* **and** *desconocido* es *desconocido*.
- **or**: el resultado de *cierto* **or** *desconocido* es *cierto*, *falso* **or** *desconocido* es *desconocido*, mientras que *desconocido* **or** *desconocido* es *desconocido*.

SQL define el resultado de una instrucción SQL de la forma

select … from R_1, …., R_n **where** *P*

para contener (proyecciones de) tuplas en $R_1 \times \ldots \times R_n$ para las que el predicado *P* se evalúa a **cierto**. Si el predicado se evalúa a **falso** o **desconocido** para una tupla de $R_1 \times \ldots \times R_n$ (la proyección de) la tupla no se añade al resultado.

SQL también permite determinar si el resultado de una comparación es desconocido en lugar de cierto o falso usando las cláusulas **is unknown** (es desconocido) e **is not unknown** (no es desconocido)

La existencia de valores nulos también complica el procesamiento de los operadores de agregación. Supóngase que algunas tuplas en la relación *préstamo* tienen valor nulo para el atributo *importe*. Considérese en ese caso la siguiente consulta, que calcula el total de todas las cantidades prestadas:

select sum (*importe*)
from *préstamo*

Los valores que van a ser sumados en la consulta anterior incluyen valores nulos, puesto que algunas tuplas tienen valor nulo para el atributo *importe*. En lugar de decir que la suma total es nula, la norma SQL establece que el operador **sum** debería ignorar los valores nulos de su entrada.

En general, las funciones de agregación tratan los valores nulos según la regla siguiente: todas las funciones de agregación excepto **count(*)** ignoran los valores nulos de la colección de datos de entrada. Como resultado de ignorar los valores nulos, la colección de valores de entrada puede resultar vacía. El cálculo de **count** de una colección vacía se define como 0 y todas las demás operaciones de agregación devuelven un valor nulo cuando se aplican sobre una colección de datos vacía. El efecto de los valores nulos en algunas de las construcciones más complicadas de SQL puede ser más sutil.

En SQL:1999 se introdujo un tipo de datos **boolean**, que puede tomar los valores **cierto**, **falso** y **desconocido**. Las funciones de agregación **some** (algún) y **every** (cada), que significan exactamente lo que se espera de ellas, se pueden aplicar a una colección de valores booleanos.

4.6. SUBCONSULTAS ANIDADAS

SQL proporciona un mecanismo para las subconsultas anidadas. Una subconsulta es una expresión **select-from-where** que se anida dentro de otra consulta. Un uso común de subconsultas es llevar a cabo comprobaciones sobre pertenencia a conjuntos, comparación de conjuntos y cardinalidad de conjuntos. Estos usos se estudiarán en los apartados siguientes.

4.6.1. Pertenencia a conjuntos

SQL utiliza el cálculo relacional para las operaciones que permiten comprobar la pertenencia de una tupla a una relación. La conectiva **in** comprueba la pertenencia a un conjunto, donde el conjunto es la colección de valores resultado de una cláusula **select**. La conectiva

not in comprueba la no pertenencia a un conjunto. Como ejemplo considérese de nuevo la consulta «Encontrar todos los clientes que tienen tanto un préstamo como una cuenta en el banco». Anteriormente escribimos esta consulta como la intersección de dos conjuntos: el conjunto de los impositores del banco y el conjunto de los prestatarios del banco. Sin embargo, existe un enfoque alternativo consistente en encontrar todos los tenedores de cuentas en el banco que son miembros del conjunto de prestatarios. Claramente, esta formulación genera el mismo resultado que la anterior, pero obliga a formular la consulta usando la conectiva **in** de SQL. A continuación, se van a obtener todos los tenedores de cuentas formulando así la siguiente subconsulta:

(**select** *nombre-cliente*
from *impositor*)

A continuación es necesario encontrar aquellos clientes que son prestatarios del banco y que aparecen en la lista de tenedores de cuenta, obtenida como resultado de la subconsulta anterior. Esto se consigue anidando la subconsulta en un **select** más externo. La consulta resultante es la siguiente:

select distinct *nombre-cliente*
from *prestatario*
where *nombre-cliente* **in** (**select** *nombre-cliente*
from *impositor*)

Este ejemplo muestra que es posible escribir la misma consulta de diversas formas en SQL. Esta flexibilidad es de gran utilidad, puesto que permite al usuario pensar en una consulta del modo que le parezca más natural. Más adelante se verá que existe una gran cantidad de redundancia en SQL.

En el ejemplo anterior se comprobaba la pertenencia a un conjunto en una relación de un solo atributo. También es posible comprobar la pertenencia a un conjunto en una relación cualquiera. Así, se puede formular la consulta «Listar los clientes que tienen tanto una cuenta como un préstamo en la sucursal Navacerrada» de un modo distinto al visto anteriormente:

select distinct *nombre-cliente*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo* =
préstamo.número-préstamo **and**
nombre-sucursal = 'Navacerrada' **and**
(*nombre-sucursal*, *nombre-cliente*) **in**
(**select** *nombre-sucursal*, *nombre-cliente*
from *impositor*, *cuenta*
where *impositor.número-cuenta*
= *cuenta. número-cuenta*)

A continuación, se ilustra el uso de la constructora **not in**. Por ejemplo, para encontrar todos los clientes que tienen un préstamo en el banco, pero no tienen una cuenta en el banco, se puede escribir

select distinct *nombre-cliente*
from *prestatario*
where *nombre-cliente* **not in** (**select** *nombre-cliente*
from *impositor*)

Los operadores **in** y **not in** también se pueden usar sobre conjuntos enumerados. La consulta siguiente selecciona los nombres de los clientes que tienen un préstamo en el banco y cuyos nombres no son ni «Santos» ni «Gómez».

select distinct *nombre-cliente*
from *prestatario*
where *nombre-cliente* **not in** ('Santos', 'Gómez')

4.6.2. Comparación de conjuntos

Considérese la consulta «Obtener los nombres de todas las sucursales que poseen un activo mayor que al menos una sucursal situada en Barcelona». En el Apartado 4.2.5 se formulaba esta consulta del modo siguiente:

select distinct *T.nombre-sucursal*
from *sucursal* **as** *T*, *sucursal* **as** S
where *T.activo* > *S.activo* **and**
S.ciudad-sucursal = 'Barcelona'

SQL ofrece, sin embargo, un estilo alternativo de formular la consulta anterior. La expresión: «mayor que al menos una» se representa en SQL por > **some**. Esta constructora permite reescribir la consulta en una forma más parecida a la formulación de la consulta en lenguaje natural.

select *nombre-sucursal*
from *sucursal*
where *activo* > **some** (**select** *activo*
from *sucursal*
where *ciudad-sucursal*
= 'Barcelona')

La subconsulta

(**select** *activo*
from *sucursal*
where *ciudad-sucursal* = 'Barcelona')

genera el conjunto de todos los valores de activo para todas las sucursales sitas en Barcelona. La comparación **> some,** en la cláusula **where** de la cláusula **select** más externa, es cierta si el valor del atributo *activo* de la tupla es mayor que al menos un miembro del conjunto de todos los valores de *activo* de las sucursales de Barcelona.

SQL también permite realizar las comparaciones **< some**, **<= some**, **>= some**, **= some** y **<> some**. Como ejer-

cicio, se puede verificar que **= some** es idéntico a **in**, mientras que **<= some** no es lo mismo que **not in**. En SQL, la palabra clave **any** es sinónimo de **some**. Las versiones más antiguas de SQL sólo admitían **any**. Sin embargo, versiones posteriores añadieron la alternativa **some** para evitar la ambigüedad lingüística de la palabra inglesa *any*.

Ahora la consulta se modificará ligeramente a fin de obtener los nombres de todas las sucursales que tienen un activo superior al de todas las sucursales de Barcelona. La constructora **> all** corresponde a la expresión «superior a todas». Utilizando esta constructora la consulta se podría formular del modo siguiente:

```
select nombre-sucursal
from sucursal
where activo > all (select activo
                    from sucursal
                    where ciudad-sucursal
                          = 'Barcelona')
```

Al igual que con **some**, SQL también permite utilizar las comparaciones **< all**, **<= all**, **>= all**, **= all** y **<> all**. Como ejercicio se puede verificar que **<= any** es lo mismo que **not in.**

Como otro ejemplo de comparaciones considérese la consulta «Encontrar la sucursal que tiene el mayor saldo medio». En SQL, las funciones de agregación no se pueden componer. Así, no está permitido el uso de **max** (**avg** (…)). Por ello, para la formulación de esta consulta se seguirá la estrategia siguiente: para empezar se formula una consulta para encontrar todos los saldos medios, y luego se anida ésta como subconsulta de una consulta más larga que encuentre aquellas sucursales para las que el saldo medio es mayor o igual que todos los saldos medios:

```
select nombre-sucursal
from cuenta
group by nombre-sucursal
having avg (saldo) >= all (select avg (saldo)
                           from cuenta
                           group by nombre-sucursal)
```

4.6.3. Comprobación de relaciones vacías

SQL incluye la posibilidad de comprobar si una subconsulta no produce ninguna tupla como resultado. La constructora **exists** devuelve el valor **cierto** si la subconsulta argumento no es vacía.

Usando la constructora **exists** se puede formular la consulta «Obtener los clientes que tienen tanto una cuenta como un préstamo en el banco» de otra nueva forma:

```
select nombre-cliente
where exists (select *
              from impositor
              where impositor.nombre-cliente =
                    prestatario.nombre-cliente)
```

Utilizando la constructora **not exists** se puede comprobar la inexistencia de tuplas en el resultado de una subconsulta. Además, es posible usar la constructora **not exists** para simular la operación de continencia de conjuntos (es decir, superconjunto). Así, se puede escribir la expresión «la relación *A* contiene a la relación *B*» como «**not exists** (B **except** A)». Aunque no forma parte de SQL estándar, el operador **contains** aparece en algunos sistemas relacionales. Para ilustrar el operador **not exists** considérese otra vez la consulta «Obtener todos los clientes que tienen una cuenta en todas las sucursales de Barcelona». Será necesario comprobar para cada cliente si el conjunto de todas las sucursales en las que dicho cliente tiene cuenta contiene al conjunto de todas las sucursales de Barcelona. Utilizando el operador **except** se puede formular la consulta del modo siguiente:

```
select distinct S.nombre-cliente
from impositor as S
where not exists ((select nombre-sucursal
                   from sucursal
                   where ciudad-sucursal
                         = 'Barcelona')
                 except
                  (select R.nombre-sucursal
                   from impositor as T, cuenta as R
                   where T.número-cuenta
                         = R.número-cuenta and
                         S.nombre-cliente
                         = T.nombre-cliente ))
```

En este ejemplo, la subconsulta

```
(select nombre-sucursal
 from sucursal
 where ciudad-sucursal = 'Barcelona')
```

obtiene todas las sucursales de Barcelona. Por otro lado, la subconsulta

```
(select R.nombre-sucursal
 from impositor as T, cuenta as R
 where T.número-cuenta = R.número-cuenta and
       S.nombre-cliente = T.nombre-cliente )
```

obtiene todas las sucursales en las cuales el cliente *S.nombre-cliente* tiene una cuenta. Por último, el **select** más externo toma cada cliente y comprueba si el conjunto de todas las sucursales en las que dicho cliente tiene cuenta, contiene al conjunto de todas las sucursales de Barcelona.

En consultas que contengan subconsultas se aplica una regla de visibilidad para las variables tupla. En una subconsulta, sólo se pueden usar variables tupla que estén definidas en la propia subconsulta o en cualquier consulta que contenga a dicha subconsulta. Si una variable tupla está definida tanto localmente (en una sub-

consulta) como globalmente (en una consulta que contenga a la subconsulta) se aplica la definición local. Esta regla es análoga a la utilizada para las variables en los lenguajes de programación.

4.6.4. Comprobación de tuplas duplicadas

SQL incluye la posibilidad de comprobar si una subconsulta produce como resultado tuplas duplicadas. La constructora **unique** devuelve el valor **cierto** si la subconsulta que se le pasa como argumento no produce tuplas duplicadas. Usando la constructora **unique** se puede formular la consulta «Obtener todos los clientes que tienen sólo una cuenta en la sucursal de nombre Navacerrada» del siguiente modo:

```
select T.nombre-cliente
from impositor as T
where unique (select R.nombre-cliente
              from cuenta, impositor as R
              where T.nombre-cliente
                    = R.nombre-cliente and
                    R.número-cuenta
                    = cuenta.número-cuenta and
                    cuenta.nombre-sucursal
                    = 'Navacerrada')
```

La existencia de tuplas duplicadas en una subconsulta se puede comprobar utilizando la constructora **not unique**. Para ilustrar esta constructora considérese la consulta «Obtener todos los clientes que tienen al menos dos cuentas en la sucursal Navacerrada», que se puede formular del modo siguiente:

```
select distinct T.nombre-cliente
from impositor as T
where not unique (select R.nombre-cliente
                  from cuenta, impositor as R
                  where T.nombre-cliente
                        = R.nombre-cliente and
                        R.número-cuenta =
                        cuenta.número-cuenta and
                        cuenta.número-sucursal
                        = 'Navacerrada')
```

Formalmente, la comprobación hecha por la constructora **unique** sobre una relación debería fallar si y sólo si en la relación existieran dos tuplas t_1 y t_2 tales que $t_1 = t_2$. Como la comprobación $t_1 = t_2$ sólo falla si cualquier campo de t_1 o de t_2 es nulo, entonces es posible que el resultado de **unique** sea **cierto** incluso si existen varias copias de una tupla, siempre que al menos uno de los atributos de la tupla sea nulo.

4.7. VISTAS

Una vista en SQL se define utilizando la orden **create view**. Para definir una vista se le debe dar un nombre y se debe construir la consulta que genere dicha vista. La forma de la orden **create view** es la siguiente:

create view *v* **as** <expresión de consulta>

donde <expresión de consulta> puede ser cualquier consulta válida. El nombre de la vista se representa por *v*. Nótese que la notación usada para la definición de una vista en el álgebra relacional (véase Capítulo 3) se basa en esta de SQL.

Como ejemplo considérese la vista consistente en los nombres de sucursales y los nombres de los clientes que tienen una cuenta o un préstamo en esa sucursal. Si se denomina esta vista como *todos-los-clientes* se definirá del modo siguiente:

```
create view todos-los-clientes as
    (select nombre-sucursal, nombre-cliente
     from impositor, cuenta
     where impositor.número-cuenta
           = cuenta.número-cuenta)
  union
    (select nombre-sucursal, nombre-cliente
     from prestatario, préstamo
     where prestatario.número-préstamo
           = préstamo.número-préstamo)
```

Los nombres de los atributos de una vista se pueden indicar explícitamente de la forma siguiente:

```
create view total-préstamos-sucursal
        (nombre-sucursal, total-préstamos) as
select nombre-sucursal, sum (importe)
from préstamo
group by nombre-sucursal
```

La vista anterior contiene para cada sucursal la suma de los importes de todos los préstamos de esa sucursal. Como la expresión **sum** (*importe*) no tiene nombre, el nombre del atributo se especifica explícitamente en la definición de la vista.

Los nombres de vistas pueden aparecer en cualquier lugar en el que pudiera aparecer un nombre de relación. Usando la vista *todos-los-clientes*, se pueden listar todos los clientes de la sucursal Navacerrada, escribiendo

```
select nombre-cliente
from todos-los-clientes
where nombre-sucursal = 'Navacerrada'
```

4.8. CONSULTAS COMPLEJAS

Las consultas complejas son a menudo difíciles o imposibles de escribir como un único bloque SQL o una unión, intersección o diferencia de bloques SQL (un bloque SQL consiste en una única instrucción **select from where**, posiblemente con cláusulas **group by** y **having**). Aquí se estudian dos formas de componer varios bloques SQL para expresar una consulta compleja: las relaciones derivadas y la cláusula **with**.

4.8.1. Relaciones derivadas

SQL permite el uso de una expresión de subconsulta en la cláusula **from**. Si se usa una expresión de este tipo se debe dar un nombre a la relación resultado y se pueden renombrar los atributos usando la cláusula **as**. Por ejemplo, considérese la subconsulta

(**select** *nombre-sucursal*, **avg** (*saldo*)
 from *cuenta*
 group by *nombre-sucursal*)
as *media-sucursal* (*nombre-sucursal*, *saldo-medio*)

Esta subconsulta produce una relación consistente en los nombres de todas las sucursales y sus correspondientes saldos de cuenta medios. El resultado de la subconsulta recibe el nombre de *media-sucursal* y contiene los atributos *nombre-sucursal* y *saldo-medio*.

Para ilustrar el uso de una expresión de subconsulta en la cláusula **from** considérese la consulta «Obtener el saldo medio de las cuentas de aquellas sucursales donde dicho saldo medio sea superior a 1.200 €». En el Apartado 4.4 se formulaba esta consulta utilizando la cláusula **having**. Ahora se puede reescribir dicha consulta sin usar esta cláusula de la siguiente forma:

select *nombre-sucursal*, *saldo-medio*
from (**select** *nombre-sucursal*, **avg** (*saldo*)
  from *cuenta*
  group by *nombre-sucursal*)
 as *resultado* (*nombre-sucursal*, *saldo-medio*)
where *saldo-medio* > 1200

En esta formulación no es necesario el uso de la cláusula **having** puesto que la relación temporal *resultado* se calcula en la cláusula **from**, y los atributos de *resultado* se pueden usar directamente en la cláusula **where**.

Supóngase como otro ejemplo que se desea hallar el máximo del total de saldos de todas las sucursales. La cláusula **having** no sirve en este caso, pero se puede escribir fácilmente esta consulta usando una subconsulta en la cláusula **from**, como se muestra a continuación:

select max(*saldo-total*)
from (**select** *nombre-sucursal*, **sum**(*saldo*)
  from *cuenta*
  group by *nombre-sucursal*) **as**
   total-sucursal(*nombre-sucursal*, *saldo-total*)

4.8.2. La cláusula with

Las consultas complicadas son mucho más fáciles de formular y de entender si se descomponen en vistas más simples y después se combinan, al igual que se estructuran los programas, descomponiendo sus tareas en procedimientos. Sin embargo, son distintas a la definición de procedimientos en cuanto a que una cláusula **create view** crea una definición de vista en la base de datos y esa definición de vista permanece en la base de datos hasta que se ejecuta una orden **drop view** *nombre-vista*.

La cláusula **with** proporciona una forma de definir una vista temporal cuya definición está disponible sólo para la consulta en la que aparece esta cláusula. Considérese la siguiente consulta, que selecciona cuentas con el saldo máximo; si hay muchas cuentas con el mismo saldo máximo, todas ellas se seleccionan.

with *saldo-máximo*(*valor*) **as**
  select max (*saldo*)
  from *cuenta*
select *número-cuenta*
from *cuenta, saldo-máximo*
where *cuenta.saldo* = *saldo-máximo.valor*

La cláusula **with** introducida en SQL:1999 se incluye actualmente sólo en algunas bases de datos.

Se podría haber escrito la consulta anterior usando una subconsulta anidada tanto en la cláusula **from** como en la **where**. Sin embargo, el uso de subconsultas anidadas hace que la consulta sea más difícil de leer y entender. La cláusula **with** hace que la lógica de la consulta sea más clara; también permite usar una definición de vista en varios lugares de una consulta.

Por ejemplo, supóngase que se desea encontrar todas las sucursales donde el depósito de cuentas es mayor que la media del total de depósitos de cuentas en todas las sucursales. Se puede escribir la consulta con la cláusula **with** como se muestra a continuación.

with *total-sucursal*(*nombre-sucursal*,*valor*) **as**
  select *nombre-sucursal*, **sum**(*saldo*)
  from *cuenta*
  group by *nombre-sucursal*
with *total-media-sucursal*(*valor*) **as**
  select avg(*sa valor ldo*)
  from *total-sucursal*

select *nombre-sucursal*
from *total-sucursal, total-media-sucursal*
where *total-sucursal.valor* > = *total-media-sucursal.valor*

Por supuesto, se puede crear una consulta equivalente sin la cláusula **with**, pero sería más complicada y difícil de entender. Como ejercicio, se puede escribir la consulta equivalente.

4.9. MODIFICACIÓN DE LA BASE DE DATOS

Hasta ahora nos hemos limitado a la extracción de información de una base de datos. A continuación se mostrará cómo añadir, eliminar o cambiar información utilizando SQL.

4.9.1. Borrado

Un borrado se expresa de igual modo que una consulta. Se pueden borrar sólo tuplas completas, es decir, no se pueden borrar valores de atributos concretos. Un borrado se expresa en SQL del modo siguiente:

delete from *r*
where *P*

donde *P* representa un predicado y *r* representa una relación. La declaración **delete** selecciona primero todas las tuplas *t* en *r* para las que *P* (*t*) es cierto y a continuación las borra de *r*. La cláusula **where** se puede omitir, en cuyo caso se borran todas las tuplas de *r*.

Hay que señalar que una orden **delete** opera sólo sobre una relación. Si se desea borrar tuplas de varias relaciones, se deberá utilizar una orden **delete** por cada relación. El predicado de la cláusula **where** puede ser tan complicado como el **where** de cualquier cláusula **select**, o tan simple como una cláusula **where** vacía. La consulta

delete from *préstamo*

borra todas las tuplas de la relación *préstamo* (los sistemas bien diseñados requerirán una confirmación del usuario antes de ejecutar una consulta tan devastadora).

A continuación se muestran una serie de ejemplos de borrados en SQL.

- Borrar todas las cuentas de la sucursal Navacerrada.

  delete from *cuenta*
  where *nombre-sucursal* = 'Navacerrada'

- Borrar todos los préstamos en los que la cantidad esté comprendida entre 1.300 € y 1.500 €.

  delete from *préstamo*
  where *importe* **between** 1300 **and** 1500

- Borrar las cuentas de todas las sucursales de Navacerrada.

  delete from *cuenta*
  where *nombre-sucursal* **in** (**select** *nombre-sucursal*
  from *sucursal*
  where *ciudad-sucursal* = 'Navacerrada')

  El borrado anterior selecciona primero todas las sucursales con sede en Navacerrada y a continuación borra todas las tuplas *cuenta* pertenecientes a esas sucursales.

Nótese que, si bien sólo se pueden borrar tuplas de una sola relación cada vez, se puede utilizar cualquier número de relaciones en una expresión **select-from-where** anidada en la cláusula **where** de un **delete**. La orden **delete** puede contener un **select** anidado que use una relación de la cual se van a borrar tuplas. Por ejemplo, para borrar todas las cuentas cuyos saldos sean inferiores a la media del banco se puede escribir:

delete from *cuenta*
where *saldo* < (**select avg** (*saldo*)
from *cuenta*)

La orden **delete** comprueba primero cada tupla de la relación *cuenta* para comprobar si la cuenta tiene un saldo inferior a la media del banco. A continuación se borran todas las tuplas que no cumplan la condición anterior, es decir, las que representan una cuenta con un saldo menor que la media. Es importante realizar todas las comprobaciones antes de llevar a cabo ningún borrado (si se borrasen algunas tuplas antes de que otras fueran comprobadas, el saldo medio podría haber cambiado y el resultado final del borrado dependería del orden en que las tuplas fueran procesadas).

4.9.2. Inserción

Para insertar datos en una relación, o bien se especifica la tupla que se desea insertar o se formula una consulta cuyo resultado sea el conjunto de tuplas que se desean insertar. Obviamente, los valores de los atributos de la tuplas que se inserten deben pertenecer al dominio de los atributos. De igual modo, las tuplas insertadas deberán ser de la aridad correcta.

La instrucción **insert** más sencilla corresponde a la de inserción de una tupla. Supongamos que se desea insertar en la base de datos el hecho de que hay una

cuenta C-9732 en la sucursal Navacerrada y que dicha cuenta tiene un saldo de 1.200 €. La inserción se puede formular del modo siguiente:

```
insert into cuenta
    values ('C-9732', 'Navacerrada', 1200)
```

En este ejemplo los valores se especifican en el mismo orden en que los atributos se listan en el esquema de relación. Para beneficio de los usuarios, que pueden no recordar el orden de los atributos, SQL permite que los atributos se especifiquen en la cláusula **insert**. Así, el siguiente ejemplo tiene una función idéntica al anterior:

```
insert into cuenta (nombre-sucursal, número-
        cuenta, saldo)
    values ('Navacerrada', 'C-9732', 1200)

insert into cuenta (número-cuenta, nombre-
        sucursal, saldo)
    values ('C-9732', 'Navacerrada', 1200)
```

Generalmente se desea insertar las tuplas que resultan de una consulta. Por ejemplo, si a todos los clientes tenedores de préstamos en la sucursal Navacerrada se les quisiera regalar, como gratificación, una cuenta de ahorro con 200 € por cada cuenta de préstamo que tienen, se podría escribir:

```
insert into cuenta
    select nombre-sucursal, número-préstamo, 200
    from préstamo
    where nombre-sucursal = 'Navacerrada'
```

En lugar de especificar una tupla, como se hizo en los primeros ejemplos de este apartado, se utiliza una instrucción **select** para especificar un conjunto de tuplas. La instrucción **select** se evalúa primero, produciendo un conjunto de tuplas que a continuación se insertan en la relación *cuenta*. Cada tupla tiene un *nombre-sucursal* (Navacerrada), un *número-préstamo* (que sirve como número de cuenta para la nueva cuenta) y un saldo inicial de la cuenta (200 €).

Además es necesario añadir tuplas a la relación *impositor*; para hacer esto, se escribirá:

```
insert into impositor
    select nombre-cliente, número-préstamo
    from prestatario, préstamo
    where prestatario.número-préstamo
            = préstamo.número-préstamo and
            nombre-sucursal = 'Navacerrada'
```

Esta consulta inserta en la relación *impositor* una tupla (*nombre-cliente*, *número-préstamo*) por cada *nombre-cliente* que posea un préstamo en la sucursal Navacerrada, con número de préstamo *número-préstamo*.

Es importante que la evaluación de la instrucción **select** finalice completamente antes de llevar a cabo ninguna inserción. Si se realizase alguna inserción antes de que finalizase la evaluación de la instrucción **select**, una consulta del tipo:

```
insert into cuenta
    select *
    from cuenta
```

podría insertar un número infinito de tuplas. La primera tupla de la relación *cuenta* se insertaría de nuevo en *cuenta*, creando así una segunda copia de la tupla. Como esta segunda copia ya sería parte de *cuenta*, la instrucción **select** podría seleccionarla, insertando así una tercera copia en la relación *cuenta*. Esta tercera copia podría ser seleccionada a continuación por el **select** e insertar una cuarta copia y así infinitamente. Evaluando completamente toda la instrucción **select** antes de realizar ninguna inserción se evitan este tipo de problemas.

Por ahora, en el estudio de la instrucción **insert** sólo se han considerado ejemplos en los que se especificaba un valor para cada atributo de las tuplas insertadas. Como se estudió en el Capítulo 3, es posible indicar sólo valores para algunos de los atributos del esquema. A cada uno de los atributos restantes, se les asignará un valor nulo, que se denota por *null*. Como ejemplo considérese la consulta:

```
insert into cuenta
    values ('C-401', null, 1200)
```

en la que se sabe que la cuenta C-401 tiene un saldo de 1.200 €, pero no se conoce el nombre de la sucursal. Considérese ahora la consulta

```
select número-cuenta
from cuenta
where nombre-sucursal = 'Navacerrada'
```

Como el nombre de la sucursal de la cuenta C-401 es desconocido, no se puede determinar si es igual a «Navacerrada».

Se puede prohibir la inserción de valores nulos utilizando el LDD de SQL, que se estudia en el Apartado 4.11.

4.9.3. Actualizaciones

En determinadas situaciones puede ser deseable cambiar un valor dentro de una tupla, sin cambiar *todos* los valores de la misma. Para este tipo de situaciones se utiliza la instrucción **update**. Al igual que ocurre con **insert** y **delete**, se pueden elegir las tuplas que van a ser actualizadas mediante una consulta.

Por ejemplo, si hubiera que realizar el pago de intereses anuales y todos los saldos se incrementasen en un 5 %, habría que formular la siguiente actualización:

```
update cuenta
set saldo = saldo * 1.05
```

Esta actualización se aplica una vez a cada tupla de la relación *cuenta*.

Si se paga el interés sólo a las cuentas con un saldo de 1.000 € o superior, se puede escribir

```
update cuenta
set saldo = saldo * 1.05
where saldo >= 1000
```

En general, la cláusula **where** de la instrucción **update** puede contener cualquier constructor legar en la cláusula **where** de una instrucción **select** (incluyendo instrucciones **select** anidadas). Como con **insert** y **delete**, un **select** anidado en una instrucción **update** puede referenciar la relación que se esté actualizando. Como antes, SQL primero comprueba todas las tuplas de la relación para determinar las que se deberían actualizar y después realiza la actualización. Por ejemplo, se puede escribir «Pagar un interés del 5% a las cuentas cuyo saldo sea mayor que la media» como sigue:

```
update cuenta
set saldo = saldo * 1.05
where (saldo > select avg(saldo)
                from cuenta)
```

Si se supone que las cuentas con saldos superiores a 10.000 € reciben un 6% de interés, mientras que las demás un 5%, se deberán escribir dos instrucciones de actualización:

```
update cuenta
set saldo = saldo * 1.06
where saldo > 10000

update cuenta
set saldo = saldo * 1.05
where saldo <= 10000
```

Obsérvese que, como se vio en el Capítulo 3, el orden en el que se ejecutan dos instrucciones de actualización es importante. Si se invierte el orden de las dos instrucciones anteriores, una cuenta con un saldo igual o muy poco inferior a 10.000 € recibiría un 11,3% de interés.

SQL ofrece una constructora **case**, que se puede usar para formular las dos instrucciones de actualización anteriores en una única instrucción de actualización, evitando el problema del orden de actualización.

```
update cuenta
set saldo = case
        when saldo <= 10000 then saldo * 1.05
        else saldo * 1.06
    end
```

La forma general de la instrucción **case** es la siguiente:

```
case
    when pred_1 then result_1
    when pred_2 then result_2
    …
    when pred_n then result_n
    else result_0
end
```

La operación devuelve $result_i$, donde i es el primero de $result_1$, $result_2$, …,$result_n$ que se satisface; si ninguno de ellos se satisface, la operación devuelve $result_0$. Las instrucciones **case** se pueden usar en cualquier lugar donde se espere un valor.

4.9.4. Actualización de vistas

La anomalía de la actualización de vistas estudiada en el Capítulo 3 también se produce en SQL. Como ejemplo considérese la siguiente definición de vista:

```
create view préstamo-sucursal as
    select nombre-sucursal, número-préstamo
    from préstamo
```

Como SQL permite que el nombre de una vista aparezca en cualquier lugar en el que pueda aparecer el nombre de una relación, se puede formular:

```
insert into préstamo-sucursal
    values ('Navacerrada', 'P-307')
```

SQL representa esta inserción mediante una inserción en la relación *préstamo*, puesto que *préstamo* es la relación real a partir de la cual se construye la vista *préstamo-sucursal*. Por lo tanto, debería especificarse un valor para el atributo *importe*. Este valor es un valor nulo. De este modo, la inserción anterior es equivalente a la inserción de la tupla

('P-307', 'Navacerrada', *null*)

en la relación *préstamo*.

Como vimos en el Capítulo 3, la anomalía de la actualización de vistas se agrava cuando una vista se define en términos de varias relaciones. Como resultado, muchas bases de datos basadas en SQL imponen la siguiente restricción a las modificaciones de vistas:

- Una modificación de una vista es válida sólo si la vista en cuestión se define en términos de la base de datos relacional real, esto es, del nivel lógico de la base de datos (y sin usar agregación).

Bajo esta restricción, las operaciones **update**, **insert** y **delete** realizadas sobre el ejemplo de la vista *todos-los-clientes* definida anteriormente, estarían prohibidas.

4.9.5. Transacciones

Una **transacción** consiste en una secuencia de instrucciones de consulta y actualizaciones. La norma SQL especifica que una transacción comienza implícitamente cuando se ejecuta una instrucción SQL. Una de las siguientes instrucciones SQL debe finalizar la transacción:

- **Commit work** compromete la transacción actual; es decir, hace que los cambios realizados por la transacción sean permanentes en la base de datos. Después de que se comprometa la transacción se inicia una nueva transacción automáticamente.
- **Rollback work** causa el retroceso de la transacción actual; es decir, deshace todas las actualizaciones realizadas por las instrucciones SQL de la transacción; así, el estado de la base de datos se restaura al que existía previo a la ejecución de la transacción.

La palabra **work** es opcional en ambas instrucciones.

El retroceso de transacciones es útil si se detecta alguna condición de error durante la ejecución de una transacción. El compromiso es similar, bajo un punto de vista, a guardar los cambios de un documento que se esté modificando, mientras que el retroceso es similar a abandonar la sesión de modificación sin guardar los cambios. Una vez que una transacción haya ejecutado **commit work**, sus efectos no se pueden deshacer con **rollback work**. El sistema de bases de datos garantiza que en el caso de una caída, los efectos de la transacción se retrocederán si no se hubo ejecutado **commit work**. En el caso de fallo de alimentación o caída del sistema, el retroceso ocurre cuando el sistema se reinicia.

Por ejemplo, para transferir dinero de una cuenta a otra es necesario actualizar dos saldos de cuenta. Las dos instrucciones de actualización formarían una transacción. Un error durante la ejecución de una de las instrucciones de una transacción resultaría en deshacer los efectos de las instrucciones anteriores de la transacción, de manera que la base de datos no se quedase en un estado parcialmente actualizado. En el Capítulo 15 se estudian más propiedades de las transacciones.

Si un programa termina sin ejecutar ninguna de estas órdenes, las actualizaciones se comprometen o retroceden. La norma no especifica cuál de los dos se debe realizar, con lo que la elección es dependiente de la implementación. En muchas implementaciones SQL, cada instrucción SQL es de manera predeterminada una transacción y se compromete tan pronto como se ejecuta. El compromiso automático de las instrucciones SQL individuales se puede desactivar si es necesario ejecutar una transacción que conste de varias instrucciones SQL. La forma de desactivación del compromiso automático depende de la implementación SQL específica.

Una alternativa mejor, que es parte de la norma SQL:1999 (pero actualmente sólo soportada por algunas implementaciones SQL), es permitir encerrar varias instrucciones SQL entre las palabras clave **begin atomic … end**. Todas las instrucciones entre las palabras clave forman así una única transacción.

4.10. REUNIÓN DE RELACIONES**

Además de proporcionar el mecanismo básico de producto cartesiano para reunir tuplas de relaciones, permitido en versiones anteriores, SQL también proporciona varios mecanismos para reunir relaciones, incluyendo reuniones condicionales y reuniones naturales, así como varias formas de reunión externa. Estas operaciones adicionales se usan a menudo como subconsultas dentro de la cláusula **from**.

4.10.1. Ejemplos

En la Figura 4.1 se muestran las relaciones *préstamo* y *prestatario* que usarán las distintas operaciones de reunión que ilustraremos. Comenzaremos con un ejemplo simple de reunión interna. En la Figura 4.2 se muestra el resultado de la expresión:

préstamo **inner join** *prestatario* **on**
préstamo.número-préstamo
= *prestatario.número-préstamo*

La expresión calcula la reunión zeta de las relaciones *préstamo* y *prestatario*, donde la condición de reunión es *préstamo.número-préstamo = prestatario.número-préstamo*. Los atributos del resultado son los atributos del lado izquierdo de la relación, seguidos por los del lado derecho de la misma.

número-préstamo	*nombre-sucursal*	*importe*
P-170	Centro	3.000
P-230	Moralzarzal	4.000
P-260	Navacerrada	1.700

préstamo

nombre-cliente	*número-préstamo*
Santos	P-170
Gómez	P-230
López	P-155

prestatario

FIGURA 4.1. Las relaciones *préstamo* y *prestatario*.

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*	*número-préstamo*
P-170	Centro	3.000	Santos	P-170
P-230	Moralzarzal	4.000	Gómez	P-230

FIGURA 4.2. Resultado de *préstamo* **inner join** *prestatario* **on** *préstamo.número-préstamo = prestatario.número-préstamo.*

Obsérvese que el atributo *número-préstamo* aparece dos veces en la figura (la primera aparición es debida a la relación *préstamo* y la segunda a *prestatario*). La norma SQL no requiere que los nombres de atributo en los resultados sean únicos. Se debería usar una cláusula **as** para asignar nombres únicos de atributos en los resultados de las consultas y subconsultas.

Renombramos la relación resultado de una reunión y los atributos de la relación resultado utilizando la cláusula **as**, como se ilustra a continuación:

préstamo **inner join** *prestatario* **on**
préstamo.número-préstamo
= *prestatario.número-préstamo*
as *ps* (*sucursal*, *número-préstamo*, *importe*,
cliente, *número-préstamo-cliente*)

La segunda aparición de *número-préstamo* se ha renombrado como *número-préstamo-cliente*. El orden de los atributos en el resultado de una reunión natural es importante a la hora de renombrarlos.

A continuación se muestra un ejemplo del uso de la operación reunión externa por la izquierda (**left outer join**):

préstamo **left outer join** *prestatario* **on**
préstamo.número-préstamo
= *prestatario.número-préstamo*

La reunión externa por la izquierda se calcula del modo siguiente. Primero se calcula el resultado de la reunión interna, como se vio anteriormente. A continuación, para cada tupla *t* de la unión interna, perteneciente a la relación del lado izquierdo (*préstamo*) que no encaje con ninguna tupla de la relación del lado derecho (*prestatario)*, se añade al resultado de la reunión una tupla *r*, como se indica a continuación. Los atributos de la tupla *r* que se derivan de la relación del lado izquierdo se rellenan con los valores de la tupla *t* y el resto de los atributos de *r* se rellenan con valores nulos. La relación resultante se muestra en la Figura 4.3. Las tuplas (P-170, Centro, 3.000) y (P-230, Moralzarzal, 4.000) se reúnen con las tuplas de *prestatario* y aparecen en el resultado de la reunión interna, y por ello en el resultado de la reunión externa por la izquierda. Por otra parte, la tupla (P-260, Navacerrada, 1.700) no encaja con ninguna tupla de *prestatario* en la reunión natural y por eso en el resultado de la reunión externa por la izquierda aparece la tupla (P-260, Navacerrada, 1.700, null, null).

Finalmente, se incluye un ejemplo del uso de la operación reunión natural (**natural join**).

préstamo **natural inner join** *prestatario*

Esta expresión calcula la reunión natural de dos relaciones. El único nombre de atributo común en *préstamo* y *prestatario* es *número-préstamo*. El resultado de la expresión anterior se muestra en la Figura 4.4. Este resultado es similar al resultado de la reunión interna con la condición **on** mostrada en la Figura 4.2, puesto que tienen la misma condición de reunión. Sin embargo, el atributo *número-préstamo* aparece sólo una vez en el resultado de la reunión natural, mientras que aparece dos veces en el resultado de la reunión con la condición **on**.

4.10.2 Tipos y condiciones de reunión

En el Apartado 4.10.1 se muestran ejemplos de los operadores de reunión incluidos en SQL-92. Las operaciones de reunión toman como entrada dos relaciones y devuelven como resultado otra relación. Aunque las expresiones de reunión externa se usan normalmente en la cláusula **from**, se pueden utilizar en cualquier lugar en el que cabría usar cualquier otra relación.

Cada variante de las operaciones de reunión en SQL-92 está formado por un *tipo de reunión* y una *condición de reunión*. La condición de reunión indica las tuplas pertenecientes a las dos relaciones que encajan y los atributos que se incluyen en el resultado de la reunión. El tipo de reunión define cómo se tratan las tuplas de cada relación que no encajan con ninguna tupla de la otra relación (basado en la condición de reunión). En la Figu-

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*	*número-préstamo*
P-170	Centro	3.000	Santos	P-170
P-230	Moralzarzal	4.000	Gómez	P-230
P-260	Navacerrada	1.700	*null*	*null*

FIGURA 4.3. Resultado de *préstamo* **left outer join** *prestatario* **on** *préstamo.número-préstamo = prestatario.número-préstamo.*

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez

FIGURA 4.4. Resultado de *préstamo* **inner join** *prestatario*.

ra 4.5 se muestran algunos de los tipos y condiciones de reunión. El primer tipo de reunión es la reunión interna y los otros tres son tres tipos de reuniones externas. Las tres condiciones de reunión son: la reunión **natural** y la condición **on**, ya vistas anteriormente, y la condición **using**, que se verá más adelante.

El uso de una condición de reunión es obligatorio en las reuniones externas, pero es opcional en las reuniones internas (ya que si se omite, el resultado será el producto cartesiano de las dos relaciones). La palabra clave **natural** aparece sintácticamente delante del tipo de reunión, como se mostró anteriormente, mientras que las condiciones **on** y **using** aparecen al final de la expresión de reunión. Las palabras clave **inner** y **outer** son opcionales, ya que el resto del tipo de reunión permite deducir si la reunión es una reunión interna o externa.

El significado de la condición de reunión **natural**, en términos de las tuplas de las relaciones que encajan, es inmediato. La ordenación de los atributos, dentro del resultado de la reunión natural es el siguiente: los atributos de reunión (es decir, los atributos comunes a las dos relaciones) aparecen en primer lugar, en el orden en el que aparezcan en la relación del lado izquierdo. A continuación están los demás atributos que no son de reunión de la relación del lado izquierdo y, al final, todos los atributos que no son de reunión de la relación del lado derecho de la relación.

El tipo de relación reunión externa por la derecha (**right outer join**) es simétrico al de reunión externa por la izquierda (**left outer join**). Las tuplas de la relación del lado derecho que no encajen con ninguna tupla de la relación del lado izquierdo se rellenan con valores nulos y se añaden al resultado de la reunión externa por la derecha.

La siguiente expresión es un ejemplo de la combinación de la condición de reunión natural con el tipo de reunión externa por la derecha.

préstamo **natural right outer join** *prestatario*

El resultado de la expresión anterior se muestra en la Figura 4.6. Los atributos del resultado se definen por el tipo de reunión, que es una reunión natural; así, *número-préstamo* aparece sólo una vez. Las primeras dos tuplas del resultado provienen de la reunión natural de *préstamo* y *prestatario*. La tupla de la relación del lado derecho (López, P-155) no encaja en la reunión interna con ninguna tupla de la relación del lado izquierdo (*préstamo*). Así, la tupla (P-155, null, López) aparece en el resultado de la reunión.

La condición de reunión **using** $(A_1, A_2, \ldots, A_n)$ es similar a la condición de reunión natural, salvo en que los atributos de reunión en este caso son $A_1, A_2, \ldots, A_n$, en lugar de todos los atributos comunes de ambas relaciones y aparecen sólo una vez en el resultado de la unión.

El tipo de reunión externa completa (**full outer join**) es una combinación de los tipos de reunión externa por la derecha y por la izquierda. Después de calcular el resultado de la reunión interna, las tuplas de la relación del lado izquierdo que no encajen con ninguna tupla de la relación del lado derecho se completan con valores nulos y se añaden al resultado. De forma análoga, las tuplas de la relación del lado derecho que no encajen con ninguna tupla de la relación del lado izquierdo, se completan con valores nulos y se añaden al resultado.

Por ejemplo, la Figura 4.7 muestra el resultado de la expresión

préstamo **full outer join** *prestatario* **using** (*número-préstamo*)

Otro ejemplo del uso de la operación reunión externa es la consulta: «Listar todos los clientes que poseen una cuenta pero no tienen un préstamo en el banco». Esta consulta se puede formular como sigue:

select *i-NC*
from (*impositor* **left outer join** *prestatario*
  on *impositor. nombre-cliente*
  = *prestatario.nombre-cliente*)
  as *db*1 (*i-NC*, *número-cuenta*, *p-NC*,
  número-préstamo)
where *p-NC* **is** *null*

De forma análoga, la consulta «Listar todos los clientes que tienen o bien una cuenta o un préstamo en el banco (pero no ambos)» podría formularse usando el operador de reunión natural externa completa, del modo siguiente:

select *nombre-cliente*
from (*impositor* **natural full outer join** *prestatario*)
where *número-cuenta* **is** *null* **or** *número-préstamo*
  is *null*

Tipos de reunión	Condiciones de reunión
inner join **left outer join** **right outer join** **full outer join**	**natural** **on** <predicado> **using** $(A_1, A_2, \ldots, A_n)$

FIGURA 4.5. Tipos y condiciones de reunión.

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez
P-155	*null*	*null*	López

FIGURA 4.6. Resultado de *préstamo* **natural right outer join** *prestatario*.

SQL-92 proporciona además otros dos tipos de reunión, llamados **cross join** (reunión cruzada) y **union join** (reunión de unión). El primero es equivalente a una reunión interna sin condición de reunión; el segundo es equivalente a una reunión externa completa con condición falsa, es decir, donde la reunión interna es vacía.

4.11. LENGUAJE DE DEFINICIÓN DE DATOS

En muchos de nuestros análisis sobre SQL y bases de datos relacionales hemos aceptado un conjunto de relaciones como predefinidas. Por supuesto, el conjunto de relaciones en una base de datos se debe especificar en términos de un lenguaje de definición de datos (LDD).

El LDD de SQL permite la especificación no sólo de un conjunto de relaciones, sino también de alguna información relativa a esas relaciones, incluyendo:

- El esquema de cada relación
- El dominio de valores asociado a cada atributo
- Las restricciones de integridad
- El conjunto de índices que se deben mantener por cada relación
- Información de seguridad y autorización para cada relación
- La estructura de almacenamiento físico de cada relación en disco.

Se analizará a continuación la definición de esquemas y de dominios de valores; el análisis de las demás características del LDD de SQL se realizará en el Capítulo 6.

4.11.1. Tipos de dominios en SQL

La norma SQL soporta un conjunto de tipos de dominios predefinidos, que incluye los siguientes:

- **char** (*n*) es una cadena de caracteres de longitud fija, con una longitud *n* especificada por el usuario. También se puede utilizar la palabra completa **character**.
- **varchar** (*n*) es una cadena de caracteres de longitud variable, con una longitud *n* especificada por el usuario. También se puede utilizar la forma completa **character varying.**.
- **int** es un entero (un subconjunto finito de los enteros, que es dependiente de la máquina). También se puede usar la palabra completa **integer**.
- **smallint** es un entero pequeño (un subconjunto del dominio de los enteros, también dependiente de la máquina).
- **numeric** (*p*,*d*) es un número en coma flotante, cuya precisión la especifica el usuario. El número está formado por *p* dígitos (más el signo), y de esos *p* dígitos, *d* pertenecen a la parte decimal. Así, **numeric** (3,1) permite que el número 44,5 se almacene exactamente, mientras que los números 444,5 y 0,32 no se pueden almacenar exactamente en un campo de este tipo.
- **real, double precision** son respectivamente números en coma flotante y números en coma flotante de doble precisión, con precisión dependiente de la máquina.
- **float** (*n*) es un número en coma flotante, cuya precisión es de al menos *n* dígitos.
- **date** es una fecha del calendario, que contiene un año (de cuatro dígitos), un mes y un día del mes.
- **time** es la hora del día, expresada en horas, minutos y segundos. Se puede usar una variante, **time**(*p*), para especificar el número de dígitos decimales para los segundos (el número predeterminado es 0). También es posible almacenar la información del uso horario junto al tiempo.
- **timestamp** es una combinación de **date** y **time**. Se puede usar una variante, **timestamp**(*p*), para

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez
P-260	Navacerrada	1.700	*null*
P-155	*null*	*null*	López

FIGURA 4.7. Resultado de *préstamo* **full outer join** *prestatario* **using** (*número-préstamo*).

especificar el número de dígitos decimales para los segundos (el número predeterminado es 6).

Los valores de fecha y hora se pueden especificar como:

date '2001-04-25'
time '09:30:00'
timestamp '2001-04-25 10:29:01.45'

Las fechas se deben especificar en el formato año seguido de mes y de día, como se muestra. El campo segundos de **time** y **timestamp** puede tener parte decimal, como se ha mostrado. Se puede usar una expresión de la forma **cast** *e* **as** *t* para convertir una cadena de caracteres (o una expresión de tipo cadena) *e* al tipo *t*, donde *t* es **date**, **time** o **timestamp**. La cadena debe estar en el formato adecuado como se indicó al comienzo de este párrafo.

Para extraer campos individuales de un valor *d* **date** o **time** se puede usar **extract**(*campo* **from** *d*), donde *campo* puede ser **year**, **month**, **day**, **hour**, **minute** o **segundo**.

Las cadenas de caracteres de longitud variable, la fecha y la hora no forman parte de la norma SQL.

SQL permite realizar operaciones de comparación sobre todos los dominios que se listan aquí, y permite realizar operaciones aritméticas y de comparación sobre los diferentes dominios numéricos. SQL también proporciona un tipo de datos llamado **interval** y permite realizar cálculos basados en fechas, horas e intervalos. Por ejemplo, si *x* e *y* son del tipo **date**, entonces *x-y* será un intervalo cuyo valor es el número de días desde la fecha *x* hasta la *y*. De forma análoga, al sumar o restar un intervalo de una fecha u hora, se obtendrá como resultado otra fecha u hora, respectivamente.

A menudo es útil poder comparar valores de dominios **compatibles**. Por ejemplo, como cada entero perteneciente al tipo **smallint** es un entero, una comparación $x < y$, donde *x* es de tipo *smallint* e *y* es de tipo **int** (o viceversa), es válida. Este tipo de comparación se lleva a cabo transformando primero el número *x* en un entero. Una transformación de este tipo se denomina *coerción*. La **coerción de tipos** se usa normalmente en los lenguajes de programación comunes, así como en los sistemas de bases de datos.

Como ejemplo, supóngase que el dominio de *nombre-cliente* es una cadena de caracteres de longitud 20 y que el dominio de *nombre-sucursal* es una cadena de caracteres de longitud 15. Aunque las longitudes de las cadenas de caracteres difieran, la norma SQL los considerará tipos compatibles.

Como se estudió en el Capítulo 3, el valor *null* pertenece a todos los dominios. Para ciertos atributos, sin embargo, los valores nulos pueden no ser apropiados. Considérese una tupla de la relación *cliente* donde el *nombre-cliente* es nulo. Una tupla de este tipo asociaría una calle y una ciudad a clientes anónimos: es decir, no contendrán información útil. En casos de este tipo, sería deseable prohibir el uso de valores nulos, restringiendo el dominio de *nombre-cliente* para excluir los valores nulos.

SQL permite incluir en la declaración de dominio de un atributo la especificación **not null** y de este modo se prohíbe la inserción de un valor nulo para ese atributo. Cualquier modificación de la base de datos que conduzca a la inserción de un valor nulo en un dominio especificado como **not null**, generará un diagnóstico de error. Existen muchas situaciones en las que sería deseable la prohibición de valores nulos. Un caso particular donde es esencial prohibir valores nulos es en la clave primaria de un esquema de relación. De este modo, en el ejemplo bancario, en la relación *cliente* se prohibirá el uso de valores nulos para el atributo *nombre-cliente*, que es la clave primaria de la relación.

4.11.2. Definición de esquemas en SQL

Un esquema de relación se define utilizando la orden **create table**:

create table r ($A_1 D_1, A_2 D_2, \ldots A_n D_n$,
⟨restricción-integridad$_1$⟩.
…
⟨restricción-integridad$_k$⟩)

donde r es el nombre de la relación, cada A_i es el nombre de un atributo del esquema de relación r y D_i es el dominio de los valores del atributo A_i. Las restricciones de integridad válidas incluyen:

- **primary key** ($A_{j_1}, A_{j_2}, \ldots, A_{j_m}$): la especificación de **clave primaria** dice que los atributos $A_{j_1}, A_{j_2}, \ldots, A_{j_m}$ forman la clave primaria de la relación. Los atributos clave primaria deben ser *no nulos* y *únicos*; es decir, ninguna tupla puede tener un valor nulo para un atributo de la clave primaria y ningún par de tuplas de la relación pueden ser iguales en todos los atributos clave primaria[1]. Aunque la especificación de clave primaria es opcional, es generalmente buena idea especificar una clave primaria para cada relación.
- **check** (*P*): la cláusula **check** especifica un predicado *P* que debe satisfacer cada tupla de la relación.

La orden de creación de tabla **create table** también incluye otras restricciones de integridad, que se estudiarán en el Capítulo 6.

En la Figura 4.8 se representa una definición parcial en SQL de la base de datos bancaria. Obsérvese que, al igual que en capítulos anteriores, no se intenta modelar con precisión el mundo real en el ejemplo de la base de

[1] En SQL-89, los atributos que forman la clave primaria no estaban declarados implícitamente como **not null**; era necesario una declaración explícita.

```
create table cliente
    (nombre-cliente   char (20),
    calle-cliente     char (30),
    ciudad-cliente    char (30),
    primary key (nombre-cliente))

create table sucursal
    (nombre-sucursal  char (15),
    ciudad-sucursal   char (30),
    activo            integer,
    primary key (nombre-sucursal),
    check (activo >= 0))

create table cuenta
    (número-cuenta    char (10),
    nombre-sucursal   char (15),
    saldo             integer,
    primary key (número-cuenta),
    check (saldo >= 0))

create table impositor
    (nombre-cliente   char (20),
    número-cuenta     char (10),
    primary key (nombre-cliente, número-cuenta))
```

FIGURA 4.8. Definición de datos en SQL para parte de la base de datos del banco.

datos bancaria. En el mundo real, muchas personas tienen el mismo nombre por lo que *nombre-cliente* no sería una clave primaria de cliente; probablemente se usaría un *id-cliente* como clave primaria. Se usa *nombre-cliente* como clave primaria para mantener el esquema de la base de datos simple y pequeño.

Si como resultado de una inserción o modificación, una tupla toma valores nulos para cualquiera de los atributos que forman parte de la clave primaria, o si tiene el mismo valor que otra tupla de la relación para éstos, SQL notifica el error y la actualización no se lleva a cabo. De forma análoga ocurre lo mismo si falla la condición **check** de una tupla.

De manera predeterminada, **null** es un valor válido para cualquier atributo en SQL, a menos que se especifique con **not null**. Un atributo se puede declarar para que no sea nulo de la forma siguiente.

número-cuenta **char**(10) **not null**

SQL también soporta una restricción de integridad

unique $(A_{j_1}, A_{j_2},\ldots,A_{j_m})$

La especificación **unique** indica que los atributos A_{j_1}, $A_{j_2},\ldots,A_{j_m}$ forman una clave candidata; es decir, no puede haber dos tuplas en la relación con todos los atributos que forman la clave candidata iguales. Sin embargo, se permite que los atributos que forman la clave candidata sean nulos, a menos que se hayan declarado como **not null**. Recuérdese que un valor nulo no es igual a ningún otro valor. El tratamiento de los valores nulos aquí es el mismo que para la constructora **unique** definida en el Apartado 4.6.4.

Un uso habitual de la cláusula **check** es el de asegurar que los valores de los atributos satisfacen unas condiciones determinadas, constituyendo así un poderoso sistema de tipos. Por ejemplo, la cláusula **check** en la orden de creación de una tabla para la relación *sucursal* comprueba que el valor para el atributo *activo* es un entero positivo. Considérese el siguiente ejemplo:

```
create table estudiante
    (nombre           char (15) not null,
    id-estudiante     char (10) not null,
    nivel-estudios    char (15) not null,
    primary key (id-estudiante),
    check (nivel-estudios in ('Graduado', 'Licenciado',
        'Doctorado' )))
```

En este ejemplo se utiliza la cláusula **check** para simular un tipo enumerado especificando que *nivel-estudios* debe ser «Graduado», «Licenciado» o «Doctorado». En el Capítulo 6 se considerarán condiciones **check** más generales, así como clases de restricciones denominadas restricciones de integridad.

Una relación inicialmente está vacía. Se pueden utilizar instrucciones de inserción para introducir datos en la misma. Muchas bases de datos relacionales tienen utilidades de carga para la introducción de un conjunto inicial de tuplas en una relación.

Para borrar una relación de una base de datos SQL, se utiliza la orden **drop table**. Dicha orden borra de la base de datos toda la información sobre la relación eliminada. La instrucción

drop table *r*

tiene una repercusión más drástica que

delete from *r*

La última conserva la relación *r*, pero borra todas sus tuplas. La primera, no sólo borra todas las tuplas de la relación *r*, sino también borra su esquema. Después de que *r* se elimine no se puede insertar ninguna tupla en dicha relación, a menos que su esquema se vuelva a crear utilizando la instrucción **create table**.

En SQL-92, la instrucción **alter table** se utiliza para añadir atributos a una relación existente. La sintaxis de la instrucción es la siguiente:

alter table *r* **add** *A D*

donde *r* es el nombre de una relación existente, *A* es el nombre del atributo que se desea añadir y *D* es el dominio del atributo *A*. Se pueden eliminar atributos de una relación utilizando la orden

alter table *r* **drop** *A*

donde *r* es el nombre de una relación existente y *A* es el nombre de un atributo de la relación. Muchos sistemas de bases de datos no permiten el borrado de atributos, aunque sí permiten el borrado de una tabla completa.

4.12. SQL INCORPORADO

SQL proporciona un lenguaje de consultas declarativo muy potente. La formulación de consultas en SQL es normalmente mucho más sencilla que la formulación de las mismas en un lenguaje de programación de propósito general. Sin embargo, el acceso a una base de datos desde un lenguaje de programación de propósito general se hace necesario al menos por dos razones:

1. No todas las consultas pueden expresarse en SQL, ya que SQL no dispone del poder expresivo de un lenguaje de propósito general. Así, existen consultas que se pueden expresar en lenguajes como Pascal, C, Cobol o Fortran y que no se pueden expresar en SQL. Para formular consultas de este tipo, podemos utilizar SQL dentro de un lenguaje más potente.

   SQL está diseñado de tal forma que las consultas formuladas puedan optimizarse automáticamente y ejecutarse de manera eficiente (al proporcionar toda la potencia de un lenguaje de programación, la optimización automática es extremadamente difícil.
2. Las acciones no declarativas (como la impresión de un informe, la interacción con un usuario o el envío de los resultados de una consulta a una interfaz gráfica) no se pueden llevar a cabo desde el propio SQL. Normalmente las aplicaciones tienen varios componentes y la consulta o actualización de datos es uno de ellos; los demás componentes se escriben en lenguajes de programación de alto nivel. En el caso de una aplicación integrada, los programas escritos en el lenguaje de programación deben tener la capacidad de acceder a la base de datos.

La norma SQL define la utilización de SQL dentro de varios lenguajes de programación, tales como C, Cobol, Pascal, Java, PL/I y Fortran. Un lenguaje en el cual se introducen consultas SQL se denomina lenguaje *anfitrión* y las estructuras SQL que se admiten en el lenguaje anfitrión constituyen SQL *incorporado*.

Los programas escritos en el lenguaje anfitrión pueden usar sintaxis SQL para acceder y actualizar datos almacenados en la base de datos. Esta forma incorporada de SQL amplía aún más la capacidad de los programadores de manipular la base de datos. En SQL incorporado, la ejecución de una consulta la realiza el sistema de base de datos y el resultado de la misma se hace disponible al programa, tupla a tupla (registro).

Un programa con SQL incorporado debe tratarse con un preprocesador especial antes de la compilación. Las consultas de SQL incorporado se sustituyen por declaraciones escritas en el lenguaje anfitrión y por llamadas a procedimientos que permiten la ejecución del acceso a la base de datos. Tras esta operación, el programa resultado se compila con el compilador del lenguaje anfitrión. Para identificar las consultas de SQL incorporado, se utiliza la instrucción EXEC SQL, que tiene la siguiente forma:

EXEC SQL <instrucción de SQL incorporado>
END-EXEC

La sintaxis exacta de las consultas en SQL incorporado depende del lenguaje dentro del que se utilicen. Por ejemplo, cuando se utilizan instrucciones de SQL dentro de un programa en C, se debe utilizar un punto y coma en lugar de END-EXEC. La incorporación de SQL en Java (denominada SQLJ) usa la sintaxis

SQL { <instrucción de SQL incorporado> };

En el programa se incluye la instrucción SQL INCLUDE para identificar el lugar donde el preprocesador debe insertar las variables especiales que se usan para la comunicación entre el programa y el sistema de base de datos. Las variables del lenguaje anfitrión se pueden utilizar en las instrucciones de SQL incorporado, pero se precederán por dos puntos (:) para distinguirlas de las variables de SQL.

Las instrucciones de SQL incorporado son similares en cuanto a la sintaxis a las instrucciones SQL que se han descrito en este capítulo. Sin embargo, hay varias diferencias que se indican a continuación.

Para formular una consulta relacional se usa la instrucción **declare cursor**. El resultado de la consulta no se calcula aún. En lugar de esto, el programa debe usar las órdenes **open** y **fetch** (que se analizarán más adelante en este apartado) para obtener las tuplas resultado.

Considerando el esquema bancario que se ha utilizado como ejemplo en este capítulo, supóngase que se tiene una variable del lenguaje anfitrión *importe* y que se desea encontrar los nombres y ciudades de residencia de aquellos clientes que superan esa cantidad en alguna de sus cuentas. Se puede escribir esta consulta del modo siguiente:

EXEC SQL
declare *c* **cursor for**
select *nombre-cliente*, *ciudad-cliente*
from *impositor*, *cliente*
where *impositor.nombre-cliente*
= *cliente.nombre-cliente* **and**
cuenta.número-cuenta
= *impositor.número-cuenta* **and**
impositor.saldo > *:importe*
END-EXEC

En la consulta anterior, la variable *c* se denomina *cursor* de la consulta. Se utiliza esta variable para identificar la consulta en la instrucción **open**, que ocasiona que se evalúe la consulta, y en la instrucción **fetch**, que permite que los valores de una tupla se obtengan como variables del lenguaje anfitrión.

La instrucción **open** para el ejemplo anterior debería ser:

EXEC SQL **open** *c* END-EXEC

Esta instrucción hace que el sistema de base de datos ejecute la consulta y guarde el resultado dentro de una relación temporal. La consulta tiene una variable del lenguaje anfitrión (:*importe*); la consulta usa el valor de la variable en el momento en que se ejecuta la instrucción **open**.

Si se produce un error como resultado de la consulta SQL, el sistema de base de datos almacena un diagnóstico de error en las variables del área de comunicación SQL (SQLCA), cuya declaración se hace mediante la instrucción SQL INCLUDE.

Un programa de SQL incorporado ejecuta una serie de instrucciones **fetch** para obtener las tuplas del resultado. La instrucción **fetch** necesita una variable del lenguaje anfitrión por cada atributo de la relación resultado. En el ejemplo anterior se necesita una variable para almacenar el valor de *nombre-cliente* y otra para el valor de *ciudad-cliente*. Si dichas variables fuesen *nc* y *cc* respectivamente, una tupla de la relación resultado se obtendría mediante la instrucción:

EXEC SQL **fetch** *c* **into** :*nc*, :*cc* END EXEC

A partir de este momento el programa puede manipular las variables *nc* y *cc*, utilizando las facilidades proporcionadas por el lenguaje anfitrión.

Un único **fetch** devuelve únicamente una tupla. Si se desean obtener todas las tuplas del resultado, el programa debe tener un bucle para iterar sobre todas las tuplas. SQL incorporado ofrece una ayuda para el programador a la hora de programar esta iteración. Aunque una relación es conceptualmente un conjunto, las tuplas resultado de una consulta están físicamente en un determinado orden fijo. Cuando se ejecuta una instrucción **open**, el cursor pasa a apuntar a la primera tupla del resultado. Al ejecutarse una instrucción **fetch**, el cursor se actualiza, pasando a apuntar a la siguiente tupla del resultado. Cuando no quedan más tuplas para ser procesadas, la variable SQLSTATE en SQLCA se establece a ‘02000’ (que significa «sin datos»). Una variable de SQLCA indica que ya no quedan tuplas por analizar en el resultado. Así, se puede utilizar un bucle **while** (o el equivalente, en función del lenguaje anfitrión) para procesar cada tupla del resultado.

La instrucción **close** se debe utilizar para indicar al sistema de base de datos que borre la relación temporal que contenía el resultado de la consulta. Para el ejemplo anterior, la sintaxis de esta instrucción será:

EXEC SQL **close** *c* END-EXEC

Las expresiones de SQL incorporado que se utilizan para modificaciones (actualización, inserción y borrado) de bases de datos no devuelven ningún resultado. Además, son más simples de expresar. Una instrucción de modificación tiene el siguiente aspecto:

EXEC SQL <cualquier actualización, inserción
o borrado válidos> END-EXEC

En la expresión de modificación pueden aparecer variables del lenguaje anfitrión precedidas de dos puntos. Si se produce un error en la ejecución de la instrucción, se devuelve un diagnóstico en SQLCA.

Las relaciones de la base de datos también se pueden actualizar con cursores. Por ejemplo, si se desea añadir 100 al atributo *saldo* de cada *cuenta* donde el nombre de sucursal sea «Navacerrada», se podría declarar un cursor como:

declare *c* **cursor for**
select *
from *account*
where *nombre-sucursal* = ‘Navacerrada’
for update

Se puede iterar por las tuplas ejecutando operaciones **fetch** sobre el cursor (como se mostró anteriormente) y después de obtener cada tupla se ejecuta el siguiente código:

update *cuenta*
set *saldo* **=** *saldo* +100
where current of *c*

SQL incorporado permite a un programa en el lenguaje anfitrión acceder a la base de datos, pero no proporciona ayuda para presentar los resultados al usuario o al generar informes. La mayoría de productos comerciales de bases de datos incluyen herramientas para ayudar a los programadores de aplicaciones a crear interfaces de usuario e informes con formato. Estas herramientas se estudian en el Capítulo 5 (Apartado 5.3).

4.13. SQL DINÁMICO

El componente *dinámico* de SQL-92 permite que en un programa se construyan y realicen consultas SQL en tiempo de ejecución. En cambio, las instrucciones de SQL incorporado deben estar presentes en tiempo de compilación y se compilan utilizando un preprocesador de SQL incorporado. Por medio de SQL dinámico los programas pueden crear consultas SQL en tiempo de ejecución (tal vez basadas en datos introducidos por el usuario) y pueden ejecutarlas inmediatamente o dejarlas *preparadas* para su ejecución. La preparación de una instrucción SQL dinámica la compila y los usos posteriores de la instrucción preparada usan la versión compilada.

SQL define normas para incorporar las llamadas de SQL dinámico dentro de lenguaje anfitrión, como C, como se muestra en el siguiente ejemplo.

```
char * prog_sql =«update cuenta set saldo
            = saldo * 1.05
            where número-cuenta = ?»
EXEC SQL prepare prog_din from :prog_sql;
char cuenta[10] = «C-101»;
EXEC SQL execute prog_din using :cuenta;
```

El programa de SQL dinámico contiene una interrogación '?' que representa una variable que se debe proporcionar en la ejecución del programa.

Sin embargo, esta sintaxis requiere extensiones para el lenguaje o un preprocesador para el lenguaje extendido. Una alternativa que usa ampliamente es una interfaz para programas de aplicación para enviar las consultas SQL o actualizaciones a un sistema de bases de datos, sin realizar cambios en el propio lenguaje de programación.

En el resto de este apartado se examinan dos normas de conexión a una base de datos SQL y la realización de consultas y actualizaciones. Una, ODBC, es una interfaz para programas de aplicación para el lenguaje C, mientras que la otra es para Java.

Para comprender estas normas es necesario comprender el concepto de sesión SQL. El usuario o aplicación se *conecta* a un servidor SQL, estableciendo una sesión; ejecuta una serie de instrucciones y, finalmente, *desconecta* la sesión. Así, todas las actividades del usuario o aplicación están en el contexto de una sesión SQL. Además de las órdenes normales de ÇSQL, una sesión también puede contener órdenes para *comprometer* el trabajo realizado en la sesión o para *retrocederlo*.

4.13.1. ODBC**

La norma **ODBC** (Open Database Connectivity, conectividad abierta de bases de datos) define una forma para que un programa de aplicación se comunique con un servidor de bases de datos. ODBC define una **interfaz para programas de aplicación** (API, Application Program Interface) que pueden usar las aplicaciones para abrir una conexión con una base de datos, enviar consultas y actualizaciones y obtener los resultados. Las aplicaciones como las interfaces gráficas de usuario, los paquetes estadísticos y las hojas de cálculo pueden usar la misma API ODBD para conectarse a cualquier servidor de bases de datos compatible con ODBC.

Cada sistema de bases de datos que sea compatible con ODBC proporciona una biblioteca que se debe enlazar con el programa cliente. Cuando el programa cliente realiza una llamada a la API ODBC, el código de la biblioteca se comunica con el servidor para realizar la acción solicitada y obtener los resultados.

La Figura 4.9 muestra un ejemplo de código C que usa la API ODBC. El primer paso al usar ODBC para comunicarse con un servidor es configurar la conexión con el servidor. Para ello, el programa asigna en primer lugar un entorno SQL, después un manejador para la conexión a la base de datos. ODBC define los tipos HENV, HDBC y RETCODE. El programa abre a continuación la conexión a la base de datos usando SQLConnect. Esta llamada tiene varios parámetros, incluyendo el manejador de la conexión, el servidor al que conectarse, el identificador de usuario y la contraseña. La constante SQL_NTS denota que el argumento anterior es una cadena terminada con nulo.

Una vez que se ha configurado la conexión, el programa puede enviar órdenes SQL a la base de datos usando SQLExecDirect. Las variables del lenguaje C se pueden vincular a los atributos del resultado de la consulta, de forma que cuando se obtenga una tupla resultado usando SQLFetch, sus valores de atributo se almacenan en las variables C correspondientes. La función SQLBindCol realiza esta tarea; el segundo argumento identifica la posición del atributo en el resultado de la consulta, y el tercer argumento indica la conversión de tipos de SQL a C requerida. El siguiente argumento da la dirección de la variable. Para los tipos de longitud variables como los arrays de caracteres, los dos últimos argumentos dan la longitud máxima de la variable y una ubicación donde almacenar la longitud actual cuando se obtenga una tupla. Un valor negativo devuelto para el campo longitud indica que el valor es **null**.

La instrucción SQLFetch está en un bucle **while** que se ejecuta hasta que SQLFetch devuelva un valor diferente de SQL_SUCCESS. En cada obtención de valores, el programa los almacena en variables C como se especifica en las llamadas en SQLBindCol e imprime estos valores.

Al final de la sesión, el programa libera el manejador, se desconecta de la base de datos y libera la conexión y los manejadores del entorno SQL. Un buen estilo de programación requiere que el resultado de cada

```
int ODBCexample()
{
    RETCODE error;
    HENV ent; /* entorno */
    HDBC con; /* conexión a la base de datos */

    SQLAllocEnv(&ent);
    SQLAllocConnect(ent, &con);
    SQLConnect(con, «aura.bell-labs.com», SQL NTS, «avi», SQL NTS, «avipasswd», SQL NTS);
    {
        char nombresucursal[80];
        float saldo;
        int lenOut1, lenOut2;
        HSTMT stmt;

        char * consulta = «select nombre_sucursal, sum (saldo)
                                from cuenta
                                group by nombre_sucursal»;
        SQLAllocStmt(con, &stmt);
        error = SQLExecDirect(stmt, consulta, SQL NTS);
        if (error == SQL SUCCESS) {
            SQLBindCol(stmt, 1, SQL C CHAR, nombresucursal, 80, &lenOut1);
            SQLBindCol(stmt, 2, SQL C FLOAT, &saldo, 0 , &lenOut2);
            while (SQLFetch(stmt) >= SQL SUCCESS) {
                printf (« %s %g\n», nombresucursal, saldo);
            }
        }
        SQLFreeStmt(stmt, SQL DROP);
    }
    SQLDisconnect(con);
    SQLFreeConnect(con);
    SQLFreeEnv(ent);
}
```

FIGURA 4.9. Código de ejemplo ODBC.

función se comprueba para asegurarse de que no haya errores; se han omitido la mayoría de estas comprobaciones por brevedad.

Es posible crear una instrucción SQL con parámetros; por ejemplo, considérese la instrucción insert into account values(?,?,?). Los interrogantes son resguardos para los valores que se proporcionarán después. Esta instrucción se puede «preparar», es decir, compilar en la base de datos y ejecutar repetidamente proporcionando los valores reales para los resguardos —en este caso proporcionando un número de cuenta, nombre de sucursal y saldo para la relación *cuenta*.

ODBC define funciones para varias tareas, tales como hallar todas las relaciones en la base de datos y los nombres y tipos de las columnas del resultado de una consulta o una relación de la base de datos.

De forma predeterminada, cada instrucción SQL se trata como una transacción separada que se compromete automáticamente. La llamada SQLSetConnectOption(con, SQL_AUTOCOMMIT, 0) desactiva el compromiso automático en la conexión con, y las transacciones se deben comprometer explícitamente con SQLTransact(con, SQL_COMMIT) o retroceder con SQLTransact(con, SQL_ROLLBACK).

Las versiones más recientes de la norma ODBC añaden nueva funcionalidad. Cada versión define *niveles de acuerdo* que especifican subconjuntos de la funcionalidad definida por el estándar. Una implementación ODBC puede proporcionar sólo las características básicas o puede proporcionar características más avanzadas (nivel 1 o 2). El nivel 1 requiere soporte para la obtención de información del catálogo, como la información sobre las relaciones existentes y los tipos de sus atributos. El nivel 2 requiere más características, como la capacidad de enviar y obtener arrays de valores de parámetros y para obtener información del catálogo más detallada.

Las normas más recientes de SQL (SQL-92 y SQL:1999) definen una **interfaz en el nivel de llamada** (Call-Level Interface, CLI) que es similar a la interfaz ODBC, pero con algunas pequeñas diferencias.

4.13.2. JDBC**

La norma **JDBC** define una API que pueden usar los programas Java para conectarse a los servidores de bases de datos (la palabra JDBC fue originalmente abreviatura de «Java Database Connectivity» —conectividad de bases de datos con Java— pero la forma completa ya no se usa). La Figura 4.10 muestra un ejemplo de un programa Java que usa la interfaz JDBC. El programa debe en primer lugar abrir una conexión a una base de datos y después ejecutar instrucciones SQL, pero antes de abrir una conexión, carga los controladores adecuados para la base de datos usando Class.forName. El primer parámetro de la llamada getConnection especifica el

```
public static void ejemploJDBC (String idbd, String idusuario, String contraseña)
{
    try
    {
        Class.forName («oracle.jdbc.driver.OracleDriver»);
        Connection con = DriverManager.getConnection
                («jdbc:oracle:thin:@aura.bell-labs.com:2000:bdbanco»,
                idusuario, contraseña);
        Statement stmt = con.createStatement();
        try {
            stmt.executeUpdate(
                    «insert into cuenta values('C-9732', 'Navacerrada', 1200)»);
        } catch (SQLException sqle)
        {
            System.out.println(«No se pudo insertar la tupla. » + sqle);
        }
        ResultSet rset = stmt.executeQuery
                («select nombre_sucursal, avg (saldo)
                from cuenta
                group by nombre_sucursal»);
        while (rset.next()) {
            System.out.println(rset.getString(«nombre_sucursal») + « » +
                    rset.getFloat(2));
        }
        stmt.close();
        con.close();
    }
    catch (SQLException sqle)
    {
        System.out.println(«SQLException : » + sqle);
    }
}
```

FIGURA 4.10. Un ejemplo de código JDBC.

nombre de la máquina en la que se ejecuta el servidor (en este caso, aura.bell-labs.com) y el número de puerto que usa para la comunicación (en este caso, 2000). El parámetro también especifica el esquema de la base de datos a usar (en este caso, bdbanco), ya que un servidor de bases de datos puede dar soporte a varios esquemas. El primer parámetro también especifica el protocolo a usar para la comunicación con la base de datos (en este caso, jdbc:oracle:thin:). Obsérvese que JDBC especifica sólo la API, no el protocolo de comunicación. Un controlador JDBC puede dar soporte a varios protocolos y se debe especificar el compatible con la base de datos y el controlador. Los otros dos argumentos de getConnection son un identificador de usuario y una contraseña.

El programa crea a continuación un manejador para la conexión y lo usa para ejecutar una instrucción SQL y obtener los resultados. En nuestro ejemplo, stmt.executeUpdate ejecuta una instrucción de actualización. El constructor try {…} catch {…} permite capturar cualquier excepción (condición de error) que surjan cuando se realizan las llamadas JDBC, e imprime un mensaje apropiado para el usuario.

El programa puede ejecutar una consulta usando stmt.executeQuery. Puede obtener el conjunto de filas en el resultado en ResultSet y leer tupla a tupla usando la función next() en el conjunto de resultados. La Figura 4.10 muestra dos formas de obtener los valores de los atributos en una tupla: usando el nombre del atributo (*nombre-sucursal*) y usando la posición del atributo (2, para denotar el segundo atributo).

También se puede crear una instrucción preparada en la que algunos valores se reemplacen por «?», especificando que los valores actuales se proporcionarán más tarde. Se pueden proporcionar los valores usando setString(). La base de datos puede compilar la consulta cuando esté preparada, y cada vez que se ejecute (con nuevos valores), la base de datos puede rehusar la forma compilada previamente de la consulta. El fragmento de código de la Figura 4.11 muestra cómo se pueden usar las instrucciones preparadas.

JDBC proporciona otras características, como los **conjuntos de resultados actualizables**. Puede crear un conjunto de resultados actualizable a partir de una consulta que realice una selección o una proyección de una

```
PreparedStatement pStmt = con.prepareStatement(
        «insert into cuenta values(?,?,?)»);
pStmt.setString(1, «C-9732»);
pStmt.setString(2, «Navacerrada»);
pStmt.setInt(3, 1200);
pStmt.executeUpdate();
pStmt.setString(1, «C-9733»);
pStmt.executeUpdate();
```

FIGURA 4.11. Instrucciones preparadas en código JDBC.

relación de la base de datos. Una actualización de una tupla en el conjunto de resultados es consecuencia de una actualización de la tupla correspondiente de la relación de la base de datos. JDBC también proporciona una API para examinar esquemas de la base de datos para encontrar los tipos de atributos de un conjunto de resultados.

Para obtener más información sobre JDBC, consúltese la información bibliográfica al final del capítulo.

4.14. OTRAS CARACTERÍSTICAS DE SQL**

El lenguaje SQL ha crecido durante las dos décadas pasadas desde un lenguaje simple con pocas características a un lenguaje ciertamente complejo con características para satisfacer a muchos tipos diferentes de usuarios. Se trataron los fundamentos de SQL anteriormente en este capítulo. En este apartado se introducen al lector algunas de las características más complejas de SQL.

4.14.1. Esquemas, catálogos y entornos

Para comprender la motivación de los esquemas y los catálogos, considérese cómo se denominan los archivos en un sistema de archivos. Los sistemas de archivos originales eran planos; es decir, todos los archivos se almacenaban en un directorio. Los sistemas de archivos de la generación actual tienen por supuesto una estructura de directorios, con archivos almacenados en subdirectorios. Para denominar unívocamente un archivo se debe especificar el nombre completo de la ruta del archivo, por ejemplo /usuarios/avi/db-book/capítulo4.tex.

Al igual que en los primeros sistemas de archivos, los primeros sistemas de bases de datos tenían un único espacio de nombres para todas las relaciones. Los usuarios tenían que coordinarse para asegurarse de que no intentaban usar el mismo nombre para relaciones diferentes. Los sistemas de bases de datos actuales proporcionan una jerarquía de tres niveles para denominar a las relaciones. El nivel superior de la jerarquía consiste en **catálogos**, cada uno de los cuales puede contener **esquemas**. Los objetos SQL tales como las relaciones y las vistas están contenidos en un **esquema**.

Para realizar cualquier acción sobre una base de datos, un usuario (o un programa) debe en primer lugar *conectarse* a la base de datos. El usuario debe proporcionar el nombre de usuario y generalmente una contraseña secreta para comprobar la identidad del usuario, como se vio en los ejemplos de ODBC y JDBC de los apartados 4.13.1 y 4.13.2. Cada usuario tiene un catálogo y esquema predeterminados, y la combinación es única para el usuario. Cuando un usuario se conecta a un sistema de bases de datos, el catálogo y esquema predeterminados se configuran para la conexión; esto se corresponde con el directorio actual establecido para el directorio inicial del usuario cuando el usuario inicia la sesión en el sistema operativo.

Para identificar una relación unívocamente, se debe usar un nombre con tres partes, por ejemplo:

catálogo5.esquema-banco.cuenta

Se puede omitir el componente catálogo y, en ese caso, la parte catálogo del nombre se considera el catálogo predeterminado para la conexión. Así, si catálogo5 es el catálogo predeterminado, se puede usar esquema-banco.cuenta para identificar la misma relación unívocamente. Además, también se puede omitir el nombre del esquema, y la parte esquema del nombre es de nuevo considerada como el esquema predeterminado de la conexión. Así, se puede usar tan solo cuenta si el catálogo predeterminado es catálogo5 y el esquema predeterminado es esquema-banco.

Con varios catálogos y esquemas disponibles pueden trabajar independientemente diferentes aplicaciones y usuarios sin preocuparse acerca de la coincidencia de nombres. Además, pueden ejecutarse varias versiones de una aplicación (una versión de producción y otras de test) en el mismo sistema de bases de datos.

El catálogo y esquema predeterminados son parte de un **entorno SQL** que se configura por cada conexión. El entorno también contiene el identificador de usuario (también conocido como *identificador de autorización*). Todas las consultas SQL habituales, incluyendo las instrucciones LDD y LMD operan en el contexto de un esquema. Los esquemas se pueden crear o eliminar mediante las instrucciones **create schema** o **drop schema**. La creación y borrado de catálogos es dependiente de la implementación y no es parte de la norma SQL.

4.14.2. Extensiones procedimentales y procedimientos almacenados

SQL proporciona un lenguaje de **módulos**, que permite definir los procedimientos en SQL. Un módulo contiene normalmente varios procedimientos SQL. Cada procedimiento tiene un nombre, parámetros opcionales y una instrucción SQL. Una extensión del lenguaje estándar SQL-92 también permite constructoras procedimentales, tales como **for**, **while** e **if-then-else**, e instrucciones SQL compuestas (varias instrucciones SQL entre **begin** y **end**).

Los procedimientos se pueden almacenar en la base de datos y ejecutarse con la instrucción **call**. Estos procedimientos se denominan también **procedimientos almacenados**. Los procedimientos almacenados son particularmente útiles porque permiten que las opera-

ciones de la base de datos se encuentren disponibles a aplicaciones externas, sin exponer ninguno de los detalles internos de la base de datos.

El Capítulo 9 trata las extensiones procedimentales de SQL, así como otras muchas características nuevas de SQL:1999.

4.15. RESUMEN

- Los sistemas de bases de datos comerciales no utilizan los lenguajes de consulta formales descritos en el Capítulo 3. El ampliamente usado lenguaje SQL, que se ha estudiado en este capítulo, está basado en el álgebra relacional formal, pero con mucho «azúcar sintáctico».
- SQL incluye varias constructoras del lenguaje para las consultas sobre la base de datos. Todas las operaciones del álgebra relacional, incluyendo las operaciones del álgebra relacional extendida, se pueden expresar en SQL. SQL también permite la ordenación de los resultados de una consulta en términos de los atributos.
- Las relaciones de vistas se pueden definir como relaciones que contienen el resultado de consultas. Las vistas son útiles para ocultar información innecesaria y para recolectar información de más de una relación en una única vista.
- Las vistas temporales definidas con la cláusula **with** también son útiles para descomponer consultas complejas en partes más pequeñas y fáciles de entender.
- SQL incluye constructoras para insertar, actualizar y borrar información. Una transacción consiste en una secuencia de operaciones que deben ser atómicas. Es decir, todas las operaciones se realizan con éxito o ninguna. En la práctica, si una transacción no se puede completar con éxito, todas las acciones parciales realizadas se deshacen.
- Las modificaciones sobre la base de datos pueden conducir a la generación de valores nulos en las tuplas. Se estudió cómo se podían introducir los valores nulos y la forma en que SQL maneja las consultas sobre las relaciones que contienen estos valores.
- El lenguaje de definición de datos SQL se usa para crear relaciones con los esquemas especificados. El LDD de SQL soporta varios tipos incluyendo **date** y **time**. Más detalles del LDD de SQL, y en particular su soporte de las restricciones de integridad, aparecen en el Capítulo 6.
- Las consultas SQL se pueden llamar desde lenguajes anfitriones mediante SQL incorporado y dinámico. Las normas ODBC y JDBC definen interfaces para programas de aplicación para acceder a bases de datos SQL desde los programas en lenguaje C y Java. Los programadores usan cada vez más estas API para acceder a bases de datos.
- También se vio una visión general de algunas características avanzadas de SQL, tales como las extensiones procedimentales, los catálogos, los esquemas y los procedimientos almacenados.

TÉRMINOS DE REPASO

- Atomicidad
- Catálogo
- Cláusula **as**
- Cláusula **from**
- Cláusula **order by**
- Cláusula **select**
- Cláusula **where**
- Cláusula **with**
- Dominios
- Duplicados
- Esquema
- Funciones de agregación
  - — **avg, min, max, sum, count**
  - — **count**
- Índice
- JDBC
- LDD: lenguaje de definición de datos
- LMD: lenguaje de manipulación de datos
- Modificación de la base de datos
  - — Actualización de vistas
  - — **delete, insert, update**
- ODBC
- Operaciones de conjuntos
  - — {<,<=,>,>=} {**some,all**}
  - — **exists**
  - — **unique**
- Operaciones de conjuntos
  - — **union, intersect, except**

- Procedimientos almacenados
- Relaciones derivadas (en la cláusula **from**)
- SQL dinámico
- SQL incorporado
- Subconsultas anidadas
- Tipos de reunión
  — **natural, using, on**
  — Reunión externa por la izquierda, por la derecha y completa
  — Reunión interna y externa
- Transacción
- Variable tupla
- Valores nulos
  — Valor «desconocido»
- Vistas

EJERCICIOS

4.1. Considérese la base de datos de seguros de la Figura 4.12, donde las claves primarias se han subrayado. Formúlense las siguientes consultas SQL para esta base de datos relacional:

a. Buscar el número total de las personas cuyos coches se han visto involucrados en un accidente en 1989.
b. Buscar el número de accidentes en los cuales se ha visto involucrado un coche perteneciente a «Santos».
c. Añadir un nuevo accidente a la base de datos; supóngase cualquier valor para los atributos necesarios.
d. Borrar el Mazda de «Santos».
e. Actualizar el importe de daños del coche de matrícula «2002BCD» en el accidente con número de informe «AR2197» a 3.000 €.

persona (id-conductor, nombre, dirección)
coche (matrícula, año, modelo)
accidente (número-informe, fecha, lugar)
es-dueño (id-conductor, matrícula)
participó (id-conductor, coche, número-informe, importe-daños)

FIGURA 4.12. Base de datos de seguros.

4.2. Considérese la base de datos de empleados de la Figura 4.13, donde las claves primarias se han subrayado. Proporciónese una expresión SQL para cada una de las consultas siguientes:

a. Buscar los nombres de todos los empleados que trabajan en el Banco Importante.
b. Buscar los nombres y ciudades de residencia de todos los empleados que trabajan en el Banco Importante.
c. Buscar los nombres, direcciones y ciudades de residencia de todos los empleados que trabajan en el Banco Importante y que ganan más de 10.000 €.
d. Buscar todos los empleados que viven en la ciudad de la empresa para la que trabajan.
e. Buscar todos los empleados que viven en la misma ciudad y en la misma calle que sus jefes.
f. Buscar todos los empleados que no trabajan en el Banco Importante.
g. Buscar todos los empleados que ganan más que cualquier empleado del Banco Pequeño.
h. Supóngase que las empresas pueden tener sede en varias ciudades. Buscar todas las empresas con sede en todas las ciudades en las que tiene sede el Banco Pequeño.
i. Buscar todos los empleados que ganan más que el sueldo medio de los empleados de su empresa.
j. Buscar la empresa que tiene el mayor número de empleados.
k. Buscar la empresa que tiene el menor sueldo medio.
l. Buscar aquellas empresas cuyos empleados ganan un sueldo más alto, en media, que el sueldo medio del Banco Importante.

empleado (nombre-empleado, calle, ciudad)
trabaja (nombre-empleado, nombre-empresa, sueldo)
empresa (nombre-empresa, ciudad)
jefe(nombre-empleado, nombre-jefe)

FIGURA 4.13. Base de datos de empleados.

4.3. Considérese la base de datos relacional de la Figura 4.13. Formúlese una expresión en SQL para cada una de las siguientes consultas:

a. Modificar la base de datos de forma que Santos viva en Ávila.
b. Incrementar en un 10% el sueldo de todos los empleados del Banco Importante.
c. Incrementar en un 10% el sueldo de todos los jefes del Banco Importante.
d. Incrementar en un 10% el sueldo de todos los empleados del Banco Importante, a menos que su sueldo pase a ser mayor de 100.000 €, en cuyo caso se incrementará su sueldo sólo en un 3%.
e. Borrar todas las tuplas de la relación *trabaja* correspondientes a los empleados del Banco Importante.

4.4. Considérense los esquemas de relación siguientes:

$$R = (A, B, C)$$
$$S = (D, E, F)$$

Además, considérense las relaciones *r (R)* y *r (S)*. Obténgase la expresión SQL equivalente a las siguientes consultas:

a. $\Pi_A(r)$
b. $\sigma_{B=17}(r)$

c. $r \times s$
d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

4.5. Sea $R = (A,B,C)$ y sean r_1 y r_2 relaciones sobre el esquema R. Proporciónese una expresión SQL equivalente a cada una de las siguientes consultas:

a. $r_1 \cup r_2$
b. $r_1 \cap r_2$
c. $r_1 - r_2$
d. $\Pi_{AB}(r_1) \bowtie \Pi_{BC}(r_2)$

4.6. Sea $R = (A,B)$ y $S=(A,C)$ y sean $r(R)$ y $s(S)$ relaciones. Formúlese una expresión SQL equivalente a cada una de las siguientes consultas:

a. $\{< a > \mid \exists\, b\, (< a, b > \in r \wedge b = 17)\}$
b. $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s\}$
c. $\{< a > \mid \exists\, c\, (< a, c > \in s \wedge \exists\, b_1, b_2\, (< a, b_1 > \in r \wedge < c, b_2 > \in r \wedge b_1 > b_2))\}$

4.7. Demuéstrese que en SQL < > **all** es equivalente a **not in**.

4.8. Considérese la base de datos relacional de la Figura 4.13. Utilizando SQL, defínase una vista que contenga *nombre-jefe* y el sueldo medio de todos los empleados que trabajan para ese jefe. Explíquese por qué el sistema de base de datos no debería permitir que las actualizaciones se expresaran en términos de esta vista.

4.9. Considérese la consulta SQL

```
select p.a1
from p, r1, r2
where p.a1 = r1.a1 or p.a1 = r2.a1
```

¿Bajo qué condiciones la consulta anterior devuelve los valores de *p.a*1 que están tanto en *r*1 como en *r*2? Examínense cuidadosamente los casos en los que *r*1.*a*1 o *r*2.*a*2 pueden ser nulos.

4.10. Escríbase una consulta SQL, sin usar la cláusula **with**, para encontrar todas las sucursales donde el depósito total de las cuentas sea menor que la media del depósito total medio en todas las sucursales usando:

a. Una consulta anidada en la cláusula **from**.
b. Una consulta anidada en una cláusula **having**.

4.11. Supóngase que se tiene una relación *nota (estudiante, puntuación)* y que se quiere clasificar a los estudiantes en función de la puntuación del modo siguiente:

SS: si la puntuación es menor que 5
AP: si la puntuación es mayor o igual que 5 y menor que 7
NT: si la puntuación es mayor o igual que 7 y menor que 8,5
SB: si la puntuación es mayor o igual que 8,5

Escríbanse consultas para hacer lo siguiente:

a. Mostrar la clasificación de cada estudiante, en términos de la relación *nota*.
b. Encontrar el número de estudiantes por clasificación.

4.12. SQL-92 proporciona una operación *n*-aria denominada **coalesce** que se define del modo siguiente: **coalesce** $(A_1, A_2, \ldots, A_n)$ devuelve el primer A_i no nulo en la lista $A_1, A_2, \ldots, A_n$ y devuelve nulo si todos ellos son nulos. Muéstrese cómo expresar la operación **coalesce** usando la operación **case**.

4.13. Sean *a* y *b* relaciones con los esquemas *A* (*nombre*, *dirección*, *puesto*) y *B* (*nombre*, *dirección*, *sueldo*), respectivamente. Indíquese cómo expresar *a* **natural full outer join** *b*, utilizando la operación **full outer join** con una condición **on** y la operación **coalesce**. Compruébese que la relación resultado no contiene dos copias de los atributos *nombre* y *dirección* y que la solución es válida incluso si dichos atributos de alguna tupla en *a* o *b* toman valor nulo para los atributos *nombre* o *dirección*.

4.14. Dada una definición de esquema SQL para la base de datos de empleados de la Figura 4.13, elíjase un dominio apropiado para cada atributo y una clave primaria para cada esquema de relación.

4.15. Escríbanse condiciones **check** para el esquema del ejercicio anterior para asegurar que:

a. Cada empleado trabaja para una empresa con sede en la ciudad de residencia del empleado.
b. Ningún empleado gana un sueldo mayor que el de su jefe.

4.16. Descríbanse las circunstancias bajo las cuales se debería utilizar SQL incorporado en lugar de SQL o un lenguaje de programación de propósito general.

NOTAS BIBLIOGRÁFICAS

Chamberlin et al. [1976] describen la versión original de SQL, denominada Sequel 2. Sequel 2 derivó de los lenguajes Square [Boyce et al., 1975] y Sequel [Chamberlin y Boyce, 1974]. La norma SQL-86 se describe en ANSI [1986]. IBM [1987] proporciona la definición de SQL de IBM System Application Architecture. Las normas oficiales de SQL-89 y de SQL-92 están disponibles en ANSI [1989] y ANSI [1992], respectivamente.

SQL-92 también se define en el U.S. Dept. of Commerce [1992] y en X/Open [1992, 1993]. Actualmente está en desarrollo la próxima versión de la norma de SQL, denominada SQL-3.

Algunos libros de texto que describen el lenguaje SQL-92 son Date y Darwen [1997], Melton y Simon [1993], y Cannan y Otten [1993]. Melton y Eisenberg [2000] proporcionan una guía de SQLJ, JDBC y tecnologías relacionadas. En http://www.sqlj.org se puede encontrar más información sobre SQLJ y software SQLJ. Date y Darwen [1997] y Date [1993a] incluyen una crítica de SQL-92.

Eisenberg y Melto [1999] proporcionan una visión general de SQL:1999. La norma está publicada como una secuencia de cinco documentos de la norma ISO/IEC, con otras partes que describen varias extensiones bajo desarrollo. La parte 1 (SQL/Framework) da una visión general de las otras partes. La parte 2 (SQL/Foundation) describe lo básico del lenguaje. La parte 3 (SQL/CLI) describe la interfaz en el nivel de llamada. La parte 4 (SQL/PSM) describe los módulos almacenados persistentes (Persistent Stored Modules, PSM), y la parte 5 (SQL/Bindings) describe los enlaces con el lenguaje anfitrión. La norma es útil para implementadores de bases de datos, pero es difícil de leer. Si se necesita, se puede comprar electrónicamente en el sitio Web http://webstore.ansi.org.

Muchos productos soportan las características de SQL además de las especificadas en las normas y muchos no soportan algunas características de la norma. Se puede encontrar más información sobre estas características en los manuales de usuario de SQL de los productos respectivos. http://java.sun.com/docs/books/tutorial es una fuente excelente para más información (actualizada) sobre JDBC y sobre Java en general. También hay disponibles en este URL referencias a libros sobre Java (incluyendo JDBC). La API ODBC se describe en Microsoft [1997] y Sanders [1998].

En los Capítulos 13 y 14 se estudia el procesamiento de consultas SQL, incluyendo algoritmos y estudios de rendimiento. También aparecen las notas bibliográficas relacionadas con este tema.

CAPÍTULO

5 OTROS LENGUAJES RELACIONALES

En el Capítulo 4 se ha descrito SQL, el lenguaje relacional de mayor influencia comercial. En este capítulo se estudiarán dos lenguajes más: QBE y Datalog. A diferencia de SQL, QBE es un lenguaje gráfico donde las consultas *parecen* tablas. QBE y sus variantes se usan ampliamente en sistemas de bases de datos para computadoras personales. Datalog tiene una sintaxis derivada del lenguaje Prolog. Aunque actualmente no se usa de forma comercial, Datalog se ha utilizado en el desarrollo de diversos sistemas de bases de datos.

En este capítulo se presentan las constructoras y conceptos fundamentales en lugar de un manual de usuario para estos lenguajes. Hay que tener presenta que las implementaciones individuales de un lenguaje puede diferir en los detalles, o puede dar soporte sólo a un subconjunto del lenguaje completo.

En este capítulo también se estudian interfaces de formularios y herramientas para generar informes y analizar datos. Aunque no son lenguajes estrictamente hablando, forman la interfaz principal a una base de datos para muchos usuarios. De hecho, la mayoría de usuarios en absoluto ejecutan consultas explícitas con un lenguaje de consulta y acceden a los datos mediante formularios, informes y otras herramientas de análisis de datos.

5.1. QUERY-BY-EXAMPLE

Query-by-Example (**QBE**, Consulta mediante ejemplos) es el nombre tanto de un lenguaje de manipulación de datos como el de un sistema de base de datos que incluyó a este lenguaje. El sistema de bases de datos QBE se desarrolló en el Centro de investigación T. J. Watson de IBM, a principios de los años setenta y el lenguaje de manipulación de datos QBE se usó más tarde en QMF (*Query Management Facility*, mecanismo de gestión de consultas), también de IBM. Actualmente, muchos de los sistemas de bases de datos para computadoras personales soportan variantes del lenguaje QBE. En este apartado se considera sólo el lenguaje de manipulación de datos. Tiene dos características distintivas:

1. A diferencia de muchos lenguajes de consulta y de programación, QBE presenta una **sintaxis bidimensional**. Las consultas *parecen* tablas. Una consulta en un lenguaje unidimensional (como SQL) se *puede* formular en una línea (posiblemente larga). Un lenguaje bidimensional *necesita* dos dimensiones para la formulación de consultas. (Existe una versión unidimensional de QBE, pero no se considerará en este estudio.)
2. Las consultas en QBE se expresan «mediante un ejemplo». En lugar de incluir un procedimiento para obtener la respuesta deseada, se usa un ejemplo de qué es lo deseado. El sistema generaliza este ejemplo para obtener la respuesta a la consulta.

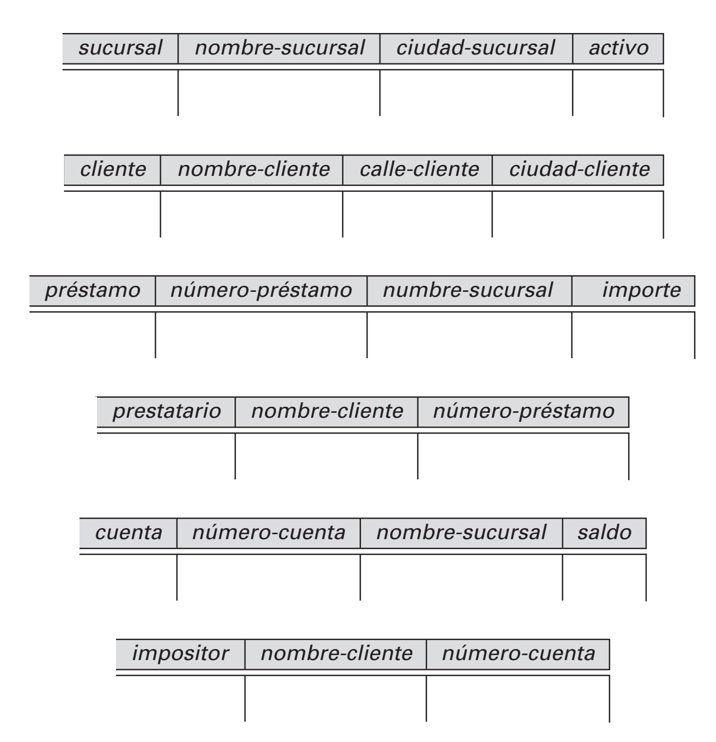
A pesar de estas características tan poco comunes existe una correspondencia entre QBE y el cálculo relacional de dominios.

Las consultas en QBE se expresan utilizando **esqueletos de tablas**. Estos esqueletos de tablas presentan el esquema de la relación, como se muestra en la Figura 5.1. En lugar de llenar la pantalla con esqueletos de tablas, el usuario elige los esqueletos que necesita para una determinada consulta y rellena dichos esqueletos **con filas ejemplo**. Una fila ejemplo está formada por constantes y *elementos ejemplo*, que son variables de dominio. Para evitar confusiones, en QBE las variables de dominio van precedidas por un carácter de subrayado (_) como en *_x*, y las constantes aparecen sin ninguna indicación particular. Este convenio contrasta con la mayoría de los lenguajes, en los que las constantes se encierran entre comillas y las variables aparecen sin ninguna indicación.

5.1.1. Consultas sobre una relación

Recuperando el ejemplo bancario que se viene utilizando en capítulos anteriores, para obtener todos los números de préstamo de la sucursal Navacerrada se utilizará el esqueleto de la relación *préstamo* y se rellenará del modo siguiente:

préstamo	*número-préstamo*	*numbre-sucursal*	*importe*
	P._*x*	Navacerrada	



sucursal	*nombre-sucursal*	*ciudad-sucursal*	*activo*

cliente	*nombre-cliente*	*calle-cliente*	*ciudad-cliente*

préstamo	*número-préstamo*	*numbre-sucursal*	*importe*

prestatario	*nombre-cliente*	*número-préstamo*

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*

impositor	*nombre-cliente*	*número-cuenta*

FIGURA 5.1. Esqueleto de las tablas QBE para el ejemplo bancario.

La consulta anterior provoca que el sistema busque tuplas en *préstamo* que tienen el atributo *nombre-sucursal* igual a «Navacerrada». Para cada tupla de este tipo, el valor del atributo *número-préstamo* se asigna a la variable *x*. El valor de la variable *x* se «imprime» (normalmente en pantalla), debido a que la orden P. aparece en la columna *número-préstamo* junto a la variable *x*. Obsérvese que este resultado es similar al que se obtendría como respuesta a la siguiente consulta del cálculo relacional de dominios:

$$\{\langle x\rangle \mid \exists\ s,c\ (\langle x, s, c\rangle \in préstamo \land s = «Navacerrada»)\}$$

QBE asume que una fila vacía contiene una variable única. Como resultado, si una variable no aparece más de una vez en una consulta, se puede omitir. La consulta anterior podría escribirse del modo siguiente:

préstamo	*número-préstamo*	*numbre-sucursal*	*importe*
	P.	Navacerrada	

QBE (a diferencia de SQL) realiza eliminación automática de duplicados. Para evitar la eliminación, se inserta la orden ALL. después de la orden P. :

préstamo	*número-préstamo*	*numbre-sucursal*	*importe*
	P.ALL.	Navacerrada	

Para mostrar la relación *préstamo* completa, se puede crear una única fila con la orden P. en todos los campos. De forma alternativa se puede usar una notación más cómoda consistente en colocar una única orden P. en la columna que lleva por nombre el nombre de la relación, es decir:

préstamo	*número-préstamo*	*numbre-sucursal*	*importe*
P.			

QBE también permite formular consultas que conlleven comparaciones aritméticas (por ejemplo >), además de las comparaciones de igualdad. Por ejemplo, «Encontrar todos los números de préstamo de aquellos préstamos con una cantidad mayor que 700 €»:

préstamo	*número-préstamo*	*numbre-sucursal*	*importe*
	P.		>700

Las comparaciones sólo pueden contener una expresión aritmética en el lado derecho de la operación de comparación (por ejemplo, $> (_x + _y - 20)$). La expresión puede contener tanto variables como constantes. El lado izquierdo de la comparación debe estar vacío. Las operaciones aritméticas que son compatibles con QBE son =, <, ≤, > ≥ y ¬.

Obsérvese que la restricción del lado derecho a una única expresión aritmética implica que no se pueden

comparar dos variables de distinto nombre (más adelante se estudiará una solución a esta dificultad).

Considérese, como otro ejemplo, la consulta «Obtener los nombres de todas las sucursales que no tienen sede en Barcelona». Esta consulta se puede formular del siguiente modo:

sucursal	nombre-sucursal	ciudad-sucursal	activo
	P.	¬ Barcelona	

La función principal de las variables en QBE es la de obligar a ciertas tuplas a tener el mismo valor en algunos atributos. Considérese la consulta «Obtener los números de préstamo de todos los préstamos pedidos conjuntamente por Santos y Gómez»:

prestatario	nombre-cliente	número-préstamo
	Santos	P._x
	Gómez	_x

Para ejecutar la consulta anterior el sistema localiza todos los pares de tuplas de la relación *prestatario* que coinciden en el atributo *número-préstamo,* donde el valor del atributo *nombre-cliente* es «Santos» para una tupla y «Gómez» para la otra. Una vez localizadas dichas tuplas, se muestra el valor del atributo *número-préstamo*.

En el cálculo relacional de dominios la consulta se podría escribir como:

$$\{\langle p \rangle \mid \exists\, x\, (\langle x, p \rangle \in prestatario \wedge x = \text{«Santos»}) \\ \wedge \exists\, x\, (\langle x, p \rangle \in prestatario \wedge x = \text{«Gómez»})\}$$

Como otro ejemplo, considérese la consulta «Obtener los nombres de todos los clientes que viven en la misma ciudad que Santos».

cliente	nombre-cliente	calle-cliente	ciudad-cliente
	P._x		_y
	Santos		_y

5.1.2. Consultas sobre varias relaciones

QBE permite formular consultas que involucren varias relaciones distintas (de igual forma que el producto cartesiano o la reunión natural en el álgebra relacional). Las conexiones entre varias relaciones se llevan a cabo a través de variables, que obligan a algunas tuplas a tomar el mismo valor en ciertos atributos. Como ejemplo, supóngase que se desea encontrar los nombres de todos los clientes que tienen un préstamo en la sucursal Navacerrada. Esta consulta se puede formular del siguiente modo:

préstamo	número-préstamo	numbre-sucursal	importe
	_x	Navacerrada	

prestatario	nombre-cliente	número-préstamo
	P._y	

Para ejecutar la consulta anterior, el sistema localiza las tuplas en la relación *préstamo* que tienen el atributo *nombre-sucursal* igual a «Navacerrada». Para cada una de esas tuplas, el sistema busca las tuplas de la relación *prestatario* con el mismo valor para el atributo *número-préstamo* que el mismo atributo de la tupla de la relación *préstamo*. Finalmente, se muestra el valor del atributo *nombre-cliente* de todas las tuplas de la relación *prestatario* que cumplan las condiciones anteriores.

Se puede utilizar una técnica similar a la anterior para formular la consulta «Encontrar los nombres de todos los clientes que tienen tanto una cuenta como un préstamo en el banco»:

impositor	nombre-cliente	número-cuenta
	P._y	

prestatario	nombre-cliente	número-préstamo
	_x	

Considérese ahora la consulta «Obtener los nombres de todos los clientes que tienen una cuenta en el banco pero que no tienen un préstamo en el mismo». En QBE las consultas que expresan negación se formulan con un signo **not** (¬) debajo del nombre de la relación y cerca de una fila ejemplo:

impositor	nombre-cliente	número-cuenta
	P._x	

prestatario	nombre-cliente	número-préstamo
¬	_x	

Compárese la consulta anterior con la formulada anteriormente: «Obtener los nombres de todos los clientes que tienen tanto una cuenta como un préstamo en el banco». La única diferencia es la aparición del ¬ en el esqueleto de la tabla *prestatario*. Esta diferencia, sin embargo, tiene un efecto más amplio en el procesamiento de la consulta. QBE busca todos los valores *x* para los cuales se cumple:

1. Existe una tupla en la relación *impositor* cuyo *nombre-cliente* es la variable de dominio *x*.
2. No existe ninguna tupla en la relación *prestatario* cuyo *nombre-cliente* es como el de la variable de dominio *x*.

El símbolo ¬ se puede interpretar como «no existe».

El hecho de que se coloque un ¬ bajo el nombre de la relación, en lugar de bajo el nombre del atributo es importante. El uso de ¬ bajo el nombre de un atributo es otro modo de expresar la condición de ≠. De este modo, para encontrar todos los clientes que tienen al menos dos cuentas se escribirá:

impositor	*nombre-cliente*	*número-cuenta*
	P._*x*	_*y*
	_*x*	¬_*y*

La consulta anterior se lee como «Mostrar todos los valores de *nombre-cliente* que aparecen al menos en dos tuplas, teniendo la segunda tupla un *número-cuenta* diferente del primero».

5.1.3. Caja de condición

Algunas veces es poco conveniente o imposible expresar todas las restricciones de las variables de dominio dentro de esqueletos de tablas. Para solucionar este problema, QBE incluye una **caja de condición**, que permite expresar restricciones generales sobre cualquiera de las variables de dominio. QBE permite que aparezcan expresiones lógicas en una caja de condición. Los operadores lógicos son las palabras **and** y **or** y los símbolos «&» y «|».

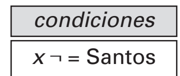
Por ejemplo, la consulta «Encontrar los números de préstamo de todos los préstamos de Santos o Gómez (o a ambos simultánemente)» se puede escribir como

prestatario	*nombre-cliente*	*número-préstamo*
	_*n*	P._*x*

condiciones
_*n* = Santos **or** _*n* Gómez

Es posible expresar esta consulta sin usar una caja de condición usando P. en varias filas. Sin embargo, las consultas con P. en varias filas son a veces difíciles de entender y es mejor evitarlas.

Como otro ejemplo, supóngase que se desea modificar la última consulta del Apartado 5.1.2 del siguiente modo: «Encontrar todos los clientes cuyo nombre no sea Santos y que tengan al menos dos cuentas en el banco». Es necesario incluir la restricción «x ≠ Santos». Para hacer esto se escribe la restricción «x ¬ = Santos» en la caja de condición:



condiciones
x ¬ = Santos

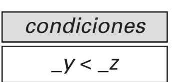
Cambiando de ejemplo, se desea obtener todos los números de cuenta con saldos entre 1.300 € y 1.500 €. Para formular esta consulta se escribirá:

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*
	P.		_*x*

condiciones
_*x* ≥ 1300
_*x* ≤ 1500

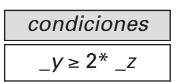
Considérese como otro ejemplo la consulta «Obtener todas las sucursales con activos superiores al activo de al menos una sucursal con sede en Barcelona». Esta consulta se formula del siguiente modo:

sucursal	*nombre-sucursal*	*ciudad-sucursal*	*activo*
	P._*x*		_*y*
		Barcelona	_*z*



condiciones
_*y* < _*z*

QBE permite la aparición de expresiones aritméticas complejas dentro de una caja de condición. Se puede formular la consulta «Encontrar todas las sucursales que tienen activos al menos dos veces mayores al activo de una de las sucursales con sede en Barcelona» del mismo modo que se formuló la consulta anterior, modificando la caja de condición:



condiciones
_*y* ≥ 2* _*z*

Para obtener todos los números de cuenta de las que tienen un saldo entre 1.300 € y 2.000 €, pero que no sea exactamente igual a 1.500 €, se escribirá:

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*
	P.		_*x*

condiciones
_*x* = (≥ 1300 **and** ≤ 2000 **and** ¬ 1500)

QBE incluye un uso poco convencional de la constructora **or** para permitir la realización de comparaciones con un conjunto de valores constantes. Para obtener todas las sucursales que tienen sede tanto en Barcelona como en Madrid, se escribirá:

sucursal	*nombre-sucursal*	*ciudad-sucursal*	*activo*
	P.	_x	

condiciones
_x = (Barcelona **or** Madrid)

5.1.4. La relación resultado

Todas las consultas que se han formulado hasta ahora tienen una característica en común: los resultados que se muestran aparecen en un único esquema de relación. Si el resultado de una consulta contiene atributos de varios esquemas de relación, se necesita un mecanismo para mostrar el resultado deseado en una única tabla. Para este propósito se puede declarar una relación temporal *resultado* que incluya todos los atributos del resultado de la consulta. Para mostrar el resultado de la consulta basta incluir la orden P. en el esqueleto de la tabla *resultado*.

Como ejemplo, considérese la consulta «Obtener el nombre de cliente, el número de cuenta y el saldo de todas las cuentas de la sucursal Navacerrada». En el álgebra relacional, esta consulta se procesará de la siguiente forma:

1. Reunión de las relaciones *impositor* y *cuenta*.
2. Proyección sobre los atributos *nombre-cliente*, *número-cuenta* y *saldo*.

Para formular esta consulta en QBE se procede del siguiente modo:

1. Se crea un esqueleto de tabla, denominado *resultado*, con los atributos *nombre-cliente*, *número-cuenta* y *saldo*. El nombre del esqueleto de tabla recién creado (es decir, *resultado*) no debe existir previamente en la base de datos como nombre de otra relación.
2. Se escribe la consulta.

La consulta resultante es:

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*
	_y	Navacerrada	_z

impositor	*nombre-cliente*	*número-cuenta*
	_x	_y

resultado	*nombre-cliente*	*número-cuenta*	*saldo*
P.	_x	_y	_z

5.1.5. Presentación ordenada de las tuplas

QBE ofrece al usuario algún control sobre el orden en el que se presentan las tuplas de una relación. Este control se expresa mediante el uso de la orden AO (ascending order, orden ascendente) y DO (descending order, orden descendente) en la columna apropiada. Así, para listar en orden alfabético ascendente todos los clientes que tienen una cuenta en el banco, escribiremos:

impositor	*nombre-cliente*	*número-cuenta*
	P.AO.	

QBE proporciona un mecanismo para la ordenación y presentación de datos en varias columnas. Para especificar el orden en el que se debe llevar a cabo la ordenación se añade a cada operador de ordenación (AO y DO) un número entero entre paréntesis. De este modo, para listar todos los números de cuenta de la sucursal Navacerrada en orden alfabético ascendente con sus respectivos saldos en orden descendente, se escribirá:

cuenta	*nombre-sucursal*	*número-cuenta*	*saldo*
	Navacerrada	P.AO(1).	P.DO(2).

La orden P.AO(1) indica que el número de cuenta se debe ordenar primero y la orden P.DO(2) indica que los saldos de cada cuenta se deben ordenar a continuación.

5.1.6. Operaciones de agregación

QBE incluye los siguientes operadores de agregación: AVG, MAX, MIN, SUM y CNT. A todos estos operadores se les debe añadir el sufijo .ALL, para crear un multiconjunto en el que se llevan a cabo las operaciones de agregación. El operador .ALL asegura que no se eliminan los duplicados. Así, para encontrar el saldo total de todas las cuentas de la sucursal Navacerrada se escribirá:

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*
		Navacerrada	P.SUM.ALL.

Se usa el operador UNQ, para especificar que se desean eliminar duplicados. Así, para encontrar el número total de clientes que tienen una cuenta en el banco se escribirá:

impositor	*nombre-cliente*	*número-cuenta*
	P.CNT.UNQ.	

QBE también ofrece la posibilidad de calcular funciones sobre grupos de tuplas, utilizando el operador G., que es análogo a la constructora **group by** de SQL. Así, para calcular el saldo medio de cada sucursal se puede escribir:

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*
		P.G.	P.AVG.ALL._*x*

El saldo medio se calcula por sucursales. La orden .ALL de la entrada P.AVG.ALL en la columna de *saldo* asegura que se están considerando todos los saldos. Si se desea mostrar los nombres de sucursal en orden ascendente se deberá sustituir P.G por P.AO.G.

Para calcular el saldo medio de las cuentas de aquellas sucursales con un saldo medio de cuenta mayor que 1.200 € se añadirá la siguiente caja de condición:

condiciones
AVG.ALL._*x* > 1200

Como último ejemplo, considérese la consulta «Obtener todos los clientes que tienen cuenta en cada una de las sucursales con sede en Barcelona»:

impositor	*nombre-cliente*	*número-cuenta*
	P.G._*x*	_*y*

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*
	_*y*	_*z*	

sucursal	*nombre-sucursal*	*ciudad-sucursal*	*activo*
	_*z*	Barcelona	
	_*w*	Barcelona	

condiciones
CNT.UNQ.ALL._*z* = CNT.UNQ.ALL._*w*

La variable de dominio *w* puede tomar el valor de nombres de sucursales con sede en Barcelona. Así, CNT.UNQ.ALL._*w* es el número de sucursales distintas de Barcelona. La variable de dominio *z* puede tomar el valor de aquellas sucursales tales que cumplen las condiciones siguientes:

- La sucursal tiene sede en Barcelona
- El cliente cuyo nombre es *x* tiene una cuenta en la sucursal.

De este modo, CNT.UNQ.ALL._*z* es el número de sucursales distintas de Barcelona en las que el cliente de nombre *x* tiene una cuenta. Si CNT.UNQ.ALL._*z* = CNT.UNQ.ALL._*w*, entonces el cliente *x* debe tener al menos una cuenta en cada una de las sucursales con sede en Barcelona. En tal caso, *x* se incluye en el resultado mostrado (debido a P.).

5.1.7. Modificaciones de la base de datos

En este apartado se mostrará cómo añadir, borrar o cambiar información utilizando QBE.

5.1.7.1. Borrado

El borrado de tuplas de una relación se expresa del mismo modo que una consulta. La diferencia principal es el uso de la orden D. en lugar de P. En QBE (a diferencia de SQL) se pueden borrar tuplas enteras, así como valores de determinadas columnas. Cuando se borra información se introducen valores nulos, denotados por –, en algunas de las columnas.

Una orden D. opera sólo sobre una relación. Si se desea borrar tuplas de varias relaciones, se debe utilizar un operador D. por cada relación.

A continuación se incluyen algunos ejemplos de borrados en QBE:

- Borrar al cliente Santos

cliente	*nombre-cliente*	*calle-cliente*	*ciudad-cliente*
D.	Santos		

- Borrar el atributo *ciudad-sucursal* de la sucursal cuyo nombre es «Navacerrada».

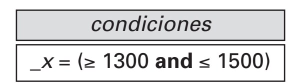
sucursal	*nombre-sucursal*	*ciudad-sucursal*	*activo*
	Navacerrada	D.	

Así, si antes de realizar la operación borrado, la relación sucursal contiene la tupla (Navacerrada, Barcelona, 50.000), el resultado consiste en el reemplazo de la tupla anterior por la de (Navacerrada, – , 50.000).

- Borrar todos los préstamos con una cantidad comprendida entre 1.300 € y 1.500 €.

préstamo	*número-préstamo*	*numbre-sucursal*	*cantidad*
D.	_*y*		_*x*

prestatario	*nombre-cliente*	*número-préstamo*
D.		_*y*



condiciones
_x = (≥ 1300 **and** ≤ 1500)

Obsérvese que para borrar préstamos se deben borrar tuplas tanto de la relación *préstamo* como de *prestatario*.

- Borrar todas las cuentas de todas las sucursales de Barcelona

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*
D.	_y	_x	

impositor	*nombre-cliente*	*número-cuenta*
D.		_y

sucursal	*nombre-sucursal*	*ciudad-sucursal*	*activo*
	_x	Barcelona	

Obsérvese que al formular un borrado se puede hacer referencia a otras relaciones además de aquéllas que tienen información sobre el borrado.

5.1.7.2. Inserción

Para insertar datos en una relación se necesita especificar la tupla que se va a ser insertar o escribir una consulta cuyo resultado sea el conjunto de tuplas que se van a insertar. La inserción se expresa mediante el operador I. Obviamente, los valores de los atributos para las tuplas insertadas deben ser miembros del dominio de los atributos.

La inserción más sencilla es la inserción de una única tupla. Supóngase que se desea insertar la cuenta C-9732 en la sucursal Navacerrada con un saldo de700 €. Se escribirá:

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*
I.	C-9732	Navacerrada	700

También se puede insertar una tupla que sólo tenga información parcial. Para insertar en la relación *sucursal* información sobre una nueva sucursal de nombre «Capital» y con sede en «Madrid», pero con un valor nulo para *activo*, se escribirá:

sucursal	*nombre-sucursal*	*ciudad-sucursal*	*activo*
I.	Capital	Madrid	

De manera más general, se puede querer insertar tuplas basadas en el resultado de una consulta. Considérese de nuevo la situación en la que se desea obsequiar a todos los tenedores de un préstamo en la sucursal Navacerrada con una nueva cuenta de ahorro con 200 € por cada cuenta de préstamo que tengan. A la nueva cuenta se le asignará el mismo número de cuenta que el número del préstamo. Se escribirá:

cuenta	*número-cuenta*	*nombre-sucursal*	*saldo*
I.	_x	Navacerrada	200

impositor	*nombre-cliente*	*número-cuenta*
I.	_y	_x

préstamo	*número-préstamo*	*numbre-sucursal*	*importe*
	_x	Navacerrada	

prestatario	*nombre-cliente*	*número-préstamo*
	_y	_x

Para ejecutar la inserción anterior el sistema debe contener la suficiente información de la relación *prestatario*. A continuación debe utilizar dicha información para insertarla apropiadamente en la nueva tupla de las relaciones *impositor* y *cuenta*.

5.1.7.3. Actualización

Existen situaciones en las cuales se desea cambiar un valor en una tupla sin cambiar todos los valores de la misma. Para este propósito se utiliza el operador U. Como ocurría con la inserción y el borrado, se pueden elegir las tuplas que se van a actualizar por medio de una consulta. QBE, sin embargo, no permite que los usuarios actualicen los campos de la clave primaria.

Supóngase que se desea actualizar el valor de activo de la sucursal Navacerrada a 10.000.000 €. Esta actualización se expresa del siguiente modo:

sucursal	*nombre-sucursal*	*ciudad-sucursal*	*activo*
	Navacerrada		U.10000000

El hecho de que el campo *ciudad-sucursal* esté vacío implica que no se realice actualización sobre este campo.

La consulta anterior actualiza el activo de la sucursal Navacerrada a 10.000.000 €, sin tener en cuenta el antiguo valor. Existen circunstancias, sin embargo, donde se necesita actualizar un valor utilizando el valor antiguo. Supóngase que se va a proceder a los pagos de inte-

reses y que todos los saldos se han de incrementar en un 5%. Se escribirá:

cuenta	número-cuenta	nombre-sucursal	saldo
			U._x * 1.05

Esta consulta indica que se recuperan de una en una las tuplas de la relación *cuenta*, se determina su saldo *x* y se actualiza dicho saldo a *x**1.05.

5.1.8. QBE de Microsoft Access

En este apartado se revisará la versión de QBE de Microsoft Access. Aunque QBE originalmente se diseñó para un entorno de visualización basado en texto, QBE de Access está diseñado para un entorno gráfico de visualización, y por lo tanto se denomina **consulta gráfica mediante ejemplos** (**GQBE**, Graphical Query-By-Example).

La Figura 5.2 muestra una consulta de ejemplo en GQBE. La consulta se puede describir en español como «Hallar *nombre-cliente, número cuenta* y *saldo* para todas las cuentas de la sucursal Navacerrada». En el Apartado 5.1.4 se mostró cómo se expresaba en QBE.

Una pequeña diferencia en la versión GQBE es que los atributos de una tabla se escriben uno debajo de otro, en lugar de horizontalmente. Una diferencia más importante es que la versión gráfica de QBE usa una línea uniendo los atributos de dos tablas, en lugar de una variable compartida, para especificar una condición de reunión (combinación, en la terminología de Microsoft).

Una característica interesante de QBE en Access es que los vínculos entre tablas se crean automáticamente en función del nombre del atributo. En el ejemplo de la Figura 5.2 se añadieron las tablas *cuenta* e *impositor*. El atributo *número-cuenta* se comparte entre las dos tablas seleccionadas y el sistema inserta automáticamente un vínculo entre las dos tablas. En otras palabras, de manera predeterminada se impone una condición de reunión natural entre las tablas; el vínculo se puede borrar si no se quiere tener. El vínculo también se puede especificar para que denote una reunión externa natural, en lugar de una reunión natural.

Otra pequeña diferencia en QBE de Access es que especifica los atributos que se mostrarán en un cuadro separado, denominada la **cuadrícula de diseño**, en lugar de usar P. en la tabla. En esta cuadrícula también especifican las selecciones según los valores de los atributos.

Las consultas que implican agrupaciones y agregaciones se pueden crear en Access como se muestra en la Figura 5.3. La consulta de la figura halla el nombre, calle y ciudad de todos los clientes que tienen más de una cuenta en el banco; se vio la versión QBE de la consulta anteriormente en el Apartado 5.1.6. Los atributos de agrupación y las funciones de agregación se anotan en la cuadrícula de diseño. Si hay que mostrar un atributo, debe aparecer en la cuadrícula de diseño, y se debe especificar en la fila «Total» que sea un criterio de agrupación o que tenga una función de agre-

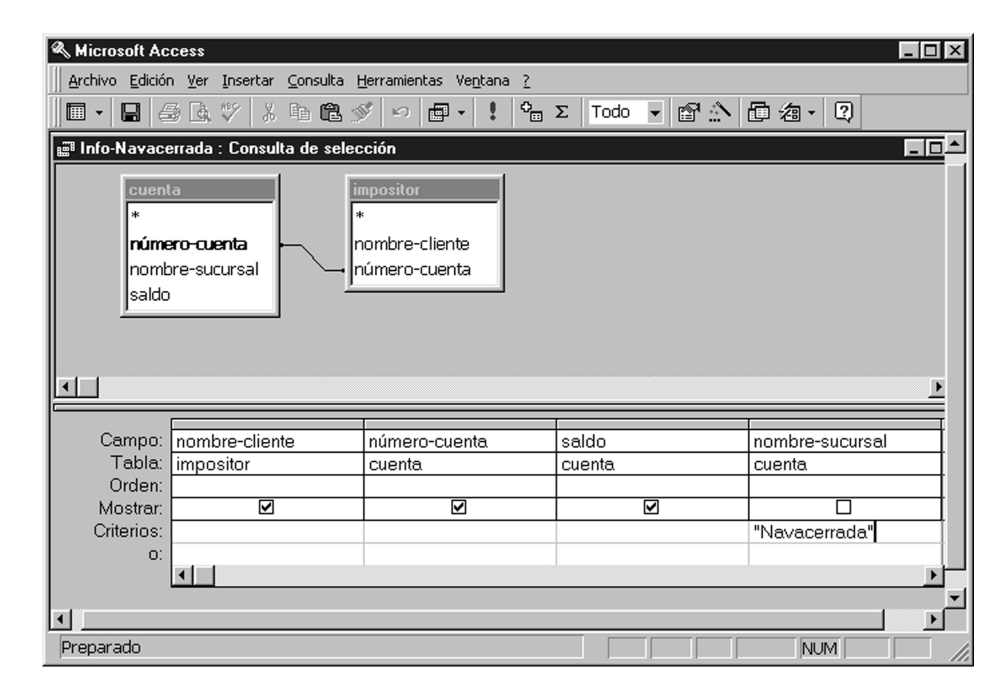


FIGURA 5.2. Una consulta de ejemplo en QBE de Microsoft Access.

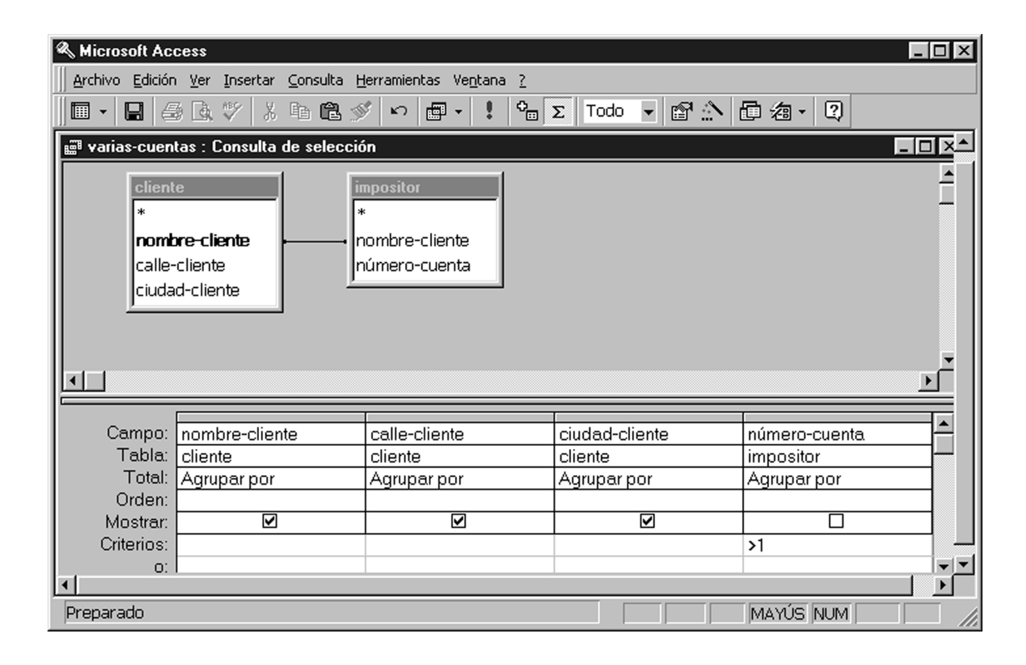


FIGURA 5.3. Una consulta de agregación en QBE de Microsoft Access.

gación aplicada. SQL tiene requisitos similares. Los atributos que participan en las condiciones de selección pero que no se muestran se pueden marcar alternativamente como «Dónde» en la fila «Total», indicando que el atributo no es ni un atributo de agrupación ni de agregación.

Las consultas se crean a través de una interfaz gráfica de usuario seleccionando en primer lugar las tablas. Los atributos se pueden añadir después a la cuadrícula de diseño arrastrándolos y soltándolos desde las tablas. Las condiciones de selección, agrupación y agregación se pueden especificar a continuación sobre los atributos de la cuadrícula de diseño. QBE de Access ofrece otra serie de características, incluyendo consultas para la modificación de la base de datos mediante inserción, borrado y actualización.

5.2. DATALOG

Datalog es un lenguaje de consultas, no procedimental, basado en el lenguaje de programación lógica Prolog. Como se hace en el cálculo relacional, el usuario describe la información deseada sin especificar un procedimiento específico de obtención de dicha información. La sintaxis de Datalog se asemeja a la de Prolog. Sin embargo, el significado de los programas en Datalog se define de una manera puramente declarativa, a diferencia de la semántica más procedimental de Prolog. Datalog simplifica la escritura de consultas simples y hace más sencilla la optimización de consultas.

5.2.1. Estructura básica

Un programa en Datalog consiste en un conjunto de **reglas**. Antes de dar una definición formal de las reglas de Datalog y su significado formal consideraremos unos cuantos ejemplos. A continuación se muestra un ejemplo de una regla Datalog para definir una relación de vistas *v1* que contiene los números de cuenta y los saldos de las cuentas de la sucursal Navacerrada cuyo saldo es superior a 700 €:

v1(*C*, *S*) :– *cuenta* (*C*, «Navacerrada», *S*), *S* > 700

Las reglas de Datalog definen vistas; la regla anterior **utiliza** la relación *cuenta* y **define** la relación de vistas *v1*. El símbolo :– se lee «si» y la coma que separa «*cuenta* (*C*, »Navacerrada«, *S*)» de «*S* > 700» se lee «y». Intuitivamente, la regla se entiende del siguiente modo:

para todo *C*,*S*
si (*C*, «Navacerrada», *S*) ∈ *cuenta* **y** *S* > 140000
entonces (*C*, *S*) ∈ *v1*

Si la relación *cuenta* es la mostrada en la Figura 5.4, entonces la vista *v1* contiene las tuplas que se muestran en la Figura 5.5.

número-cuenta	*nombre-sucursal*	*saldo*
C-101	Centro	500
C-215	Becerril	700
C-102	Navacerrada	400
C-305	Collado Mediano	350
C-201	Navacerrada	900
C-222	Moralzarzal	700
C-217	Navacerrada	750

FIGURA 5.4. La relación *cuenta*.

Para calcular el saldo de la cuenta C-217 en la vista *v1* se puede formular la consulta siguiente:

? *v1*(«C-217», *S*)

El resultado de la consulta anterior es:

(C-217, 750)

Para obtener el número y el saldo de todas las cuentas de la vista *v1* con saldo superior a 800 €, se puede escribir:

? *v1*(*C*,*S*), *S* > 160000

El resultado de la consulta anterior es

(C-201, 900)

En general puede ser necesaria más de una regla para definir una vista. Cada regla define un conjunto de tuplas que debe contener la vista. Así, el conjunto de tuplas de la vista se define como la unión de todos esos conjuntos de tuplas. El siguiente programa Datalog especifica los tipos de interés para las cuentas.

tipo-interés (*C*,5) :– *cuenta* (*C, N, S*), *S* < 10000
tipo-interés (*C*,6) :– *cuenta* (*C, N, S*), *S* > = 10000

El programa tiene dos reglas que definen la vista *tipo-interés*, cuyos atributos son el número de cuenta y el tipo de interés. Las reglas especifican que si el saldo es menor que 10.000 €, el tipo de interés es 5% y si el saldo es igual o superior a 10.000 €, entonces el tipo de interés es el 6%.

Las reglas Datalog pueden utilizar la negación. Las reglas siguientes definen una vista *c*, que contiene los nombres de todos los clientes que tienen cuenta, pero no tienen ningún préstamo en el banco:

c(*N*) :– *impositor* (*N*,*C*) , **not** *es-prestatario* (*N*)
es-prestatario (*N*) :– *prestatario* (*N*,*P*)

número-cuenta	*saldo*
C-201	900
C-217	750

FIGURA 5.5. La relación *v1*.

En Prolog, y en la mayoría de las implementaciones de Datalog, los atributos de una relación se reconocen por su posición y el nombre de los atributos se omite. Así, las reglas de Datalog son compactas en comparación con las consultas de SQL. Sin embargo, cuando las relaciones tienen un gran número de atributos o el orden de los atributos de una relación puede variar, la notación posicional puede conducir a errores. No es difícil crear una sintaxis de Datalog que reconozca los atributos por el nombre en lugar de por la posición. En un sistema de este tipo, la regla Datalog que define *v1* se podría escribir del modo siguiente:

v1(*número-cuenta C*, *saldo S*) :–
cuenta (*número-cuenta C*, *nombre-sucursal* «Navacerrada», *saldo S*),
S > 700

La transición entre las dos variantes no implica un esfuerzo significativo disponiendo del esquema de relación.

5.2.2. Sintaxis de las reglas de Datalog

Una vez que se han explicado informalmente las reglas y consultas en Datalog, se define formalmente su sintaxis. Se utilizarán los mismos convenios que en el álgebra relacional para denotar nombres de relaciones, de atributos, de constantes (tales como números o cadenas) y de nombres de variables. Se usan letras mayúsculas y palabras con la primera letra en mayúscula para denotar nombres de variables, y letras minúsculas y palabras con la primera letra en minúscula para denotar los nombres de las relaciones y atributos. Algunos ejemplos de constantes son 4, que es un número, y «Santos», que es una cadena; *X* y *Nombre* son variables. Un **literal positivo** tiene la siguiente forma:

$$p\ (t_1, t_2, \dots, t_n)$$

donde *p* es el nombre de una relación con *n* atributos y $t_1, t_2, \dots, t_n$ son constantes o variables. Un **literal negativo** tiene la siguiente forma:

$$\textbf{not}\ p\ (t_1, t_2, \dots, t_n)$$

donde la relación *p* tiene *n* atributos. El siguiente es un ejemplo de literal:

cuenta (*C*, «Navacerrada», *S*)

Los literales que contienen operaciones aritméticas se tratan de un modo especial. Por ejemplo, el literal *B* > 700, aunque no tiene la sintaxis descrita, puede entenderse conceptualmente como > (B, 700), que *sí* tiene la sintaxis requerida y donde > es una relación.

Pero ¿qué significa esta notación para las operaciones aritméticas como «>»? La relación > (conceptual-

mente) contiene tuplas de la forma (x,y) para todos los posibles pares de valores x e y donde $x > y$. Así, (2, 1) y (5, –33) son tuplas de la relación >. La relación > es infinita. Otras operaciones aritméticas (como >, =, + o –) se tratan también conceptualmente como relaciones. Por ejemplo, A = B + C se puede tratar conceptualmente como + (B, C, A), donde la relación + contiene todas las tuplas (x, y, z) tales que $z = x + y$.

Un **hecho** tiene la siguiente forma:

$$p\ (v_1, v_2, \ldots, v_n)$$

e implica que la tupla $(v_1, v_2, \ldots, v_n)$ pertenece a la relación p. Un conjunto de hechos de una relación se puede escribir utilizando la notación tabular habitual. Un conjunto de hechos para una relación en un esquema de base de datos equivale a un ejemplar del esquema de base de datos. Las **reglas** se construyen a partir de literales, y tienen la forma siguiente:

$$p\ (t_1, t_2, \ldots, t_n)\ :\!-\ L_1, L_2, \ldots, L_n$$

donde cada L_i es un literal (positivo o negativo). El literal $p(t_1, t_2, \ldots, t_n)$ se denomina **cabeza** de la regla y el resto de los literales constituyen el **cuerpo** de la misma.

Un **programa Datalog** consiste en un conjunto de reglas; el orden en el que se formulan las reglas es indiferente. Como se mencionó anteriormente, una relación se puede definir utilizando varias reglas.

En la Figura 5.6 se muestra un ejemplo de un programa Datalog relativamente complejo, que define el tipo de interés para cada cuenta de la sucursal Navacerrada. La primera regla del programa define una vista *interés*, cuyos atributos son el número de cuenta y el interés de la cuenta. Usa la relación *cuenta* y la vista *tipo-interés*. Las últimas dos reglas del programa son reglas que ya se vieron anteriormente.

Una vista v_1 se dice que **depende directamente de** una vista v_2 si v_2 se usa en la expresión que define a v_1. En el programa, la vista *tipo-interés* depende directamente de las relaciones *tipo-interés* y *cuenta*. A su vez, la relación *tipo-interés* depende directamente de cuenta.

Una vista v_1 se dice que **depende indirectamente de** una vista v_2 si hay una secuencia de relaciones intermedias $i_1, i_2, \ldots, i_n$ para algún n tal que v_1 depende directamente de i_1, i_1 depende directamente de i_2, y así sucesivamente hasta i_{n-1} que depende directamente de i_n.

```
interés (C, I) :– cuenta (C, «Navacerrada», S),
                  tipo-interés (C, T), I = S * T / 100.
tipo-interés (C, 5) :– cuenta (C, N, S), S < 10000
tipo-interés (C, 6) :– cuenta (C, N, S), S >= 10000
```

FIGURA 5.6. Programa Datalog que define el interés de las cuentas de la sucursal Navacerrada.

En el ejemplo de la Figura 5.6, dado que hay una cadena de dependencias entre *interés* y *tipo-interés* hasta *cuenta*, la relación *interés* depende indirectamente de *cuenta*.

Finalmente, se dice que una vista v_1 **depende de** una vista v_2 si v_1 depende directa o indirectamente de v_2.

Se dice que una vista v es **recursiva** si depende de sí misma. Una vista que no depende de sí misma se dice que **no es recursiva**.

Considérese el programa de la Figura 5.7. En él, la vista *empl* depende de sí misma (debido a la segunda regla) y por tanto es recursiva. En cambio, el programa de la Figura 5.6 no es recursivo.

5.2.3. Semántica de Datalog no recursivo

A continuación se considera la semántica formal de los programas Datalog. Por ahora se analizarán únicamente programas no recursivos. La semántica de programas recursivos es algo más complicada; se analizará en el Apartado 5.2.6. La semántica de un programa se define empezando por la semántica de una regla.

5.2.3.1. Semántica de una regla

Un **ejemplar básico de una regla** es el resultado de sustituir cada variable de la regla por alguna constante. Si una variable aparece varias veces en una regla, todas las apariciones de la variable se deben sustituir por la misma constante. Los ejemplares básicos se llaman habitualmente **ejemplares**.

A continuación se muestra el ejemplo de definición de la vista *v1* y un ejemplar de la regla:

v1 (*C*,*S*) :– *cuenta* (*C*, «Navacerrada», *S*) , *S* > 700
v1 («C-217» , 750) :– *cuenta* («C-217», «Navacerrada», 750), 750 > 700

En este ejemplo la variable *C* ha sido sustituida por «C-217» y la variable *S* por 750.

Normalmente una regla tiene muchos ejemplares posibles. Estos ejemplares se corresponden con las diversas formas de asignar valores a cada variable de la regla.

Dada la regla *R* siguiente:

$$p\ (t_1, t_2, \ldots, t_n)\ :\!-\ L_1, L_2, \ldots, L_n$$

y un conjunto de hechos *I* asociados a las relaciones que aparecen en la regla. Considérese cualquier ejemplar *R*′ de la regla *R*:

$$p\ (v_1, v_2, \ldots, v_n)\ :\!-\ l_1, l_2, \ldots, l_n$$

```
empl (X,Y) :– jefe (X, Y)
empl (X,Y) :– jefe (X, Z), empl (Z, Y)
```

FIGURA 5.7. Programa Datalog recursivo.

donde cada literal l_i es de la forma $q_i\ (v_{i,1}, v_{i,2}, \ldots, v_{i,n_i})$ o **not** $q_i\ (v_{i,1}, v_{i,2}, \ldots, v_{i,n_i})$ y donde cada v_i y cada $v_{i,j}$ es una constante.

Se dice que el cuerpo de un ejemplar R' se **satisface** en I si

1. Para cada literal positivo $q_i\ (v_{i,1}, v_{i,2}, \ldots, v_{i,n_i})$ del cuerpo de R', el conjunto de hechos I contiene el hecho $q_i\ (v_{i,1}, v_{i,2}, \ldots, v_{i,n_i})$.
2. Para cada literal negativo **not** $q_j\ (v_{j,1}, v_{j,2}, \ldots, v_{j,n_j})$ del cuerpo de R', el conjunto de hechos I no contiene el hecho $q_j\ (v_{j,1}, v_{j,2}, \ldots, v_{j,n_j})$.

Se define el conjunto de hechos que se pueden **inferir** a partir de un conjunto de hechos I dado usando la regla R como:

$$\textit{inferir}\ (R, I) = \{p\ (t_1, t_2, \ldots, t_{n_i}) \mid \text{existe un ejemplar } R' \text{ de } R, \text{ donde } p\ (t_1, t_2, \ldots, t_{n_i}) \text{ es la cabeza de } R' \text{ y el cuerpo de } R' \text{ se satisface en } I\}$$

Dado un conjunto de reglas $R = \{R_1, R_2, \ldots, R_n\}$ se define:

$$\textit{inferir}\ (R, I) = \textit{inferir}\ (R_1, I) \cup \textit{inferir}\ (R_2, I) \cup \ldots \cup \textit{inferir}\ (R_n, I)$$

Dado un conjunto de hechos I, que contiene las tuplas de la relación *cuenta* que se muestra en la Figura 5.4, un posible ejemplar de la regla ejemplo R sería:

v1 («C-217», 750) :– *cuenta* («C-217», «Navacerrada», 750) , 750 > 700

El hecho *cuenta* («C-217», «Navacerrada», 750) pertenece al conjunto de hechos I. Además, 750 es mayor que 700 y así, conceptualmente, (750, 700) está en la relación «>». Por tanto, el cuerpo del ejemplar de la regla se satisface en I. Existen otros posibles ejemplares de R y utilizándolos se comprueba que *inferir* (R, I) tiene exactamente el conjunto de hechos para *v1* que se muestra en la Figura 5.8.

5.2.3.2. Semántica de un programa

Cuando una vista se define en términos de otra vista, el conjunto de hechos de la primera vista depende de los hechos de la segunda de ellas. En este apartado se asume que las definiciones no son recursivas: es decir, ninguna vista depende (directa o indirectamente) de sí misma. Así, se pueden superponer las relaciones de vistas en capas de la forma siguiente y se puede usar la superposición para definir la semántica del programa.

- Una relación está en la capa 1 si todas las relaciones que aparecen en los cuerpos de las reglas que la definen están almacenadas en la base de datos.
- Una relación está en la capa 2 si todas las relaciones que aparecen en los cuerpos de las reglas que la definen están almacenadas en la base de datos, o son de la capa 1.
- En general, una relación p está en la capa $i + 1$ si (1) no está en las capas 1, 2, …, i, y (2) todas las relaciones que aparecen en los cuerpos de las reglas que la definen están almacenadas en la base de datos o son de las capas 1, 2, …, i.

Considérese el programa de la Figura 5.6. La clasificación en capas de las vistas del programa se muestra en la Figura 5.9. La relación *cuenta* está en la base de datos. La relación *tipo-interés* está en la capa 1, mientras que todas las relaciones que se utilizan en las dos reglas que la definen están en la base de datos. La relación *cuenta-Navacerrada* está también en la capa 1. Por último, la relación interés está en la capa 2, puesto que no está en la capa 1 y todas las relaciones que se utilizan en las reglas que la definen están en la base de datos o en capas inferiores a la 2.

A continuación se puede definir la semántica de un programa Datalog en términos de la clasificación en capas de las vistas. Sean las capas de un programa dado en el rango 1,2,…,n. Sea Ri el conjunto de todas las reglas que definen vistas en la capa i.

- Se define I_0 como el conjunto de los hechos almacenados en la base de datos e I_1 como

$$I_1 = I_0 \cup \textit{inferir}\ (R_1\ , I_0)$$

- A continuación se procede de un modo análogo, definiendo I_2 en términos de I_1 y R_2, y así sucesivamente utilizando la siguiente definición:

$$I_{i+1} = I_i \cup \textit{inferir}\ (R_{i+1}\ , I_i)$$

número-cuenta	*saldo*
C-201	900
C-217	750

FIGURA 5.8. Resultado de *inferir* (*R, I*).

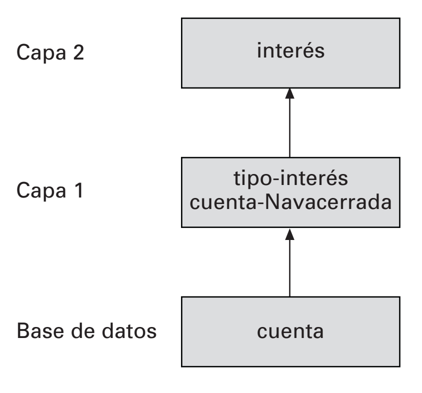


FIGURA 5.9. Clasificación en capas de las vistas.

- Por último, el conjunto de hechos de las vistas definidos por el programa (también denominados **la semántica del programa**) se define como el conjunto de hechos I_n, correspondientes a la capa más alta (n).

Para el programa de la Figura 5.6, I_0 es el conjunto de hechos en la base de datos e I_1 es el conjunto de hechos de la base de datos ampliado con el conjunto de todos los hechos que se pueden inferir de I_0 utilizando las reglas de la relación *tipo-interés* y *cuenta-Navacerrada*. Finalmente, I_2 contiene los hechos de I_1, junto con los hechos de la relación *interés* que se pueden inferir de los hechos en I_1 utilizando las reglas de la relación *interés*. La semántica del programa, es decir, el conjunto de los hechos que están en cada vista, se define como el conjunto de los hechos de I_2.

Recuérdese que en el Apartado 3.5.3 se mostró cómo definir el significado de las vistas no recursivas del álgebra relacional utilizando una técnica denominada expansión de vistas. La expansión de vistas se puede usar también con vistas no recursivas de Datalog; del mismo modo, la técnica de clasificación por capas descrita anteriormente se puede utilizar con las vistas del álgebra relacional.

5.2.4. Seguridad

Es posible formular reglas que generen un infinito número de respuestas. Considérese la regla

$$mayor\ (X, Y)\ \ :\!-\ X > Y$$

Como la relación que define > es infinita, esta regla generaría un número infinito de hechos para la relación *mayor*, cuyo cálculo necesitaría una cantidad infinita de tiempo y espacio.

El uso de la negación puede causar problemas similares. Considérese la regla siguiente:

no-en-préstamo (*S, P*) :– **not** *préstamo* (*S, P*)

La idea es que la tupla (*sucursal*, *número-préstamo*) está en la vista *no-en-préstamo* si no pertenece a la relación *préstamo*. Sin embargo, si el conjunto de posibles nombres de sucursales o el número de cuentas es infinito, la relación *no-en-préstamo* puede ser infinita.

Por último, si existe una variable en la cabeza de la regla que no aparece en el cuerpo, se puede generar un conjunto infinito de hechos donde la variable se asigna a distintos valores.

Teniendo en cuenta que estas posibilidades se pueden producir, se exige que las reglas de Datalog cumplan las siguientes condiciones de **seguridad**:

1. Cualquier variable que aparezca en el lado izquierdo de una regla debe aparecer en un literal positivo no aritmético, en el cuerpo de la misma.
2. Cualquier variable que aparezca en un literal negativo en la cabeza de una regla debe aparecer también en algún literal positivo en el cuerpo de la misma.

Si todas las reglas de un programa Datalog no recursivo satisfacen las reglas anteriores entonces las vistas definidas en el programa serán finitas, siempre que las relaciones de la base de datos sean finitas. Las condiciones se pueden relajar en algunos casos para permitir que variables de la cabeza puedan aparecer sólo en literales aritméticos del cuerpo. Por ejemplo, en la regla

$$p\ (A)\ \ :\!-\ q\ (B),\ A = B + 1$$

se puede observar que si la relación q es finita, entonces, por las propiedades de la adición, también lo es p, incluso cuando la variable A aparece sólo en un literal aritmético.

5.2.5. Operaciones relacionales en Datalog

Las expresiones de Datalog no recursivas sin operaciones aritméticas son equivalentes en poder expresivo a las expresiones que utilizan las operaciones básicas del álgebra relacional (∪, – , ×, σ, Π y ρ). En lugar de demostrar formalmente ahora esta afirmación, se mostrará mediante ejemplos cómo se pueden expresar en Datalog varias operaciones del álgebra relacional. En todos los casos se define una vista denominada *consulta*, para ilustrar las operaciones.

Ya se ha visto anteriormente cómo llevar a cabo una selección utilizando reglas de Datalog. Las proyecciones se llevan a cabo utilizando únicamente los atributos requeridos en la parte izquierda de la regla. Para proyectar el atributo *nombre-cuenta* de la relación *cuenta*, se utilizará:

$$consulta\ (C)\ \ :\!-\ cuenta\ (C, N, S)$$

Para obtener el producto cartesiano de dos relaciones r_1 y r_2 en Datalog, se puede formular la regla:

$$consulta\ (X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_m)\ \ :\!-\ r_1\ (X_1, X_2, \ldots, X_n),\ r_2\ (Y_1, Y_2, \ldots, Y_m)$$

Donde r_1 es de aridad n, r_2 es de aridad m y los $X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_m$ son nombres de variables, todos distintos.

La unión de dos relaciones r_1 y r_2 (ambas de aridad n), se forma del siguiente modo:

$$consulta\ (X_1, X_2, \ldots, X_n)\ \ :\!-\ r_1\ (X_1, X_2, \ldots, X_n)$$
$$consulta\ (X_1, X_2, \ldots, X_n)\ \ :\!-\ r_2\ (X_1, X_2, \ldots, X_n)$$

El conjunto diferencia de dos relaciones r_1 y r_2 se calcula como sigue:

$$consulta\ (X_1, X_2, \ldots, X_n)\ :\!-\ r_1\ (X_1, X_2, \ldots, X_n),\ \textbf{not}\ r_2\ (X_1, X_2, \ldots, X_n)$$

Finalmente, obsérvese que, con la notación posicional usada en Datalog, el operador renombramiento ρ no se necesita. Una relación puede aparecer más de una vez en el cuerpo de la regla, pero en lugar de renombrar para dar nombres diferentes a las apariciones de la relación, se usan diferentes nombres de variables en las diferentes apariciones.

Es posible demostrar que cualquier consulta no recursiva de Datalog se puede expresar sin funciones aritméticas, utilizando las operaciones del álgebra relacional. Esta demostración se plantea como ejercicio para el lector. Así, se puede establecer la equivalencia de las operaciones básicas del álgebra relacional y Datalog no recursivo sin operaciones aritméticas.

Las operaciones relacionales extendidas de inserción, borrado y actualización son compatibles con ciertas extensiones de Datalog. La sintaxis para tales operaciones varía entre implementaciones distintas. Algunos sistemas permiten el uso de + o – en la parte izquierda de las reglas para denotar la inserción y borrado relacional. Por ejemplo, se pueden trasladar todas las cuentas de la sucursal Navacerrada a la sucursal Madrid ejecutando

+ *cuenta* (*C*, «Madrid», *S*) :– *cuenta* (*C*, «Navacerrada», *S*)
– *cuenta* (*C*, «Navacerrada», *S*) :– *cuenta* (*C*, «Navacerrada», *S*)

La operación de agregación del álgebra relacional extendida se ofrece en algunas implementaciones de Datalog. De nuevo, no existe sintaxis normalizada para esta operación.

5.2.6. La recursividad en Datalog

Diversas aplicaciones de base de datos manejan estructuras similares a los árboles de datos. Por ejemplo, considérense los empleados de una empresa. Algunos empleados son jefes. Cada jefe dirige un conjunto de personas y cada una de esas personas puede ser a su vez jefe. Así, los empleados se pueden organizar en una estructura similar a un árbol.

Supóngase el siguiente esquema de relación:

Esquema-jefe = (*nombre-empleado, nombre-jefe*)

Sea *jefe* una relación del esquema anterior.

Supóngase ahora que se desea obtener un listado de los empleados que supervisa (directa o indirectamente) un determinado jefe, por ejemplo Santos. Es decir, si el jefe de Arteaga es Benzoaga, el jefe de Benzoaga es Erejalde y el jefe de Erejalde es Santos, entonces Arteaga, Benzoaga y Erejalde son empleados supervisados por Santos. A menudo se escriben programas recursivos para manipular árboles de datos. Utilizando la idea de la recursividad se puede definir el conjunto de personas dirigidas por Santos como se indica a continuación. Las personas supervisadas por Santos son (1) personas cuyo jefe directo es Santos y (2) las personas cuyo jefe es supervisado por Santos. Nótese que el caso (2) es recursivo.

```
procedure PuntoFijo-Datalog
    I = conjunto de hechos de la base de datos
    repeat
        I_Anterior = I
        I = ∪ inferir (R, I)
    until I = I_Anterior
```

FIGURA 5.10. Procedimiento PuntoFijo-Datalog.

Se puede formular la definición recursiva anterior como una vista recursiva de Datalog, denominada *empl-Santos*, del modo siguiente:

empl-Santos (*X*) :– *jefe* (*X*,«Santos»)
empl-Santos (*X*) :– *jefe* (*X*,*Y*), *empl-Santos* (*Y*)

La primera regla corresponde al caso (1) y la segunda al caso (2). La vista *empl-Santos* depende de sí misma por la segunda regla; por eso, el programa Datalog anterior es recursivo. Se *asumirá* a partir de ahora que los programas de Datalog recursivos no contienen reglas con literales negativos. La razón se aclarará más adelante. Las notas bibliográficas hacen referencia a publicaciones que describen dónde se puede utilizar la negación en programas Datalog.

Las vistas de un programa recursivo que contienen un conjunto de reglas *R* se definen exactamente como el conjunto de hechos *I* que resultan de aplicar iterativamente el procedimiento PuntoFijo-Datalog de la Figura 5.10. La recursividad en un programa Datalog se convierte en la iteración del procedimiento. Al final del procedimiento aparece *inferir* (*R*, *I*) ∪ *D* = *I*, donde *D* es el conjunto de hechos de la base de datos, e *I* se denomina **punto fijo** del programa.

Considérese el programa que define *empl-Santos* de la Figura 5.11 con la relación *jefe*. En la Figura 5.12 se muestra el conjunto de hechos que resultan de cada iteración de la vista *empl-Santos*. En cada iteración, se calcula un nivel más de empleados dirigidos por Santos, que se añaden al conjunto *empl-Santos*. El procedi-

nombre-empleado	*nombre-jefe*
Arteaga	Benzoaga
Benzoaga	Erejalde
Conesa	Duarte
Duarte	Santos
Erejalde	Santos
Santos	Lara
Ruipérez	Lara

FIGURA 5.11. La relación *jefe*.

número de Iteración	Tuplas en empl-Santos
0	
1	(Duarte), (Erejalde)
2	(Duarte), (Erejalde), (Benzoaga), (Conesa)
3	(Duarte), (Erejalde), (Benzoaga), (Conesa), (Arteaga)
4	(Duarte), (Erejalde), (Benzoaga), (Conesa), (Arteaga)

FIGURA 5.12. Empleados de Santos en las distintas iteraciones del procedimiento PuntoFijo-Datalog.

miento termina cuando, tras una iteración, no se produce ningún cambio en el conjunto *empl-Santos*, es decir, cuando *I = I_Anterior*. Como el conjunto de empleados y jefes es finito, en algún momento se tiene que alcanzar este punto fijo. En la relación *jefe* dada, el procedimiento PuntoFijo-Datalog termina después de la cuarta iteración, al detectar que no se ha inferido ningún hecho nuevo.

Se debe verificar que, al final de cada iteración, la vista *empl-Santos* contiene aquellos empleados que trabajan bajo la supervisión de Santos. Para obtener los nombres de los empleados supervisados por Santos, se puede utilizar la siguiente consulta:

? empl-Santos (*N*)

Para que se entienda mejor el procedimiento PuntoFijo-Datalog hay que señalar que una regla infiere nuevos hechos a partir de un conjunto de hechos dado. La iteración comienza con un conjunto de hechos *I* que corresponden a los hechos de la base de datos. Estos hechos se sabe a ciencia cierta que son verdaderos pero pueden existir otros hechos igualmente verdaderos[1]. A continuación, se aplica el conjunto de reglas *R* del programa Datalog, partiendo del conjunto de hechos verdaderos *I*. Los hechos inferidos se añaden a *I* y las reglas se usan de nuevo para hacer nuevas inferencias. Este proceso se repite hasta que no se infieren nuevos hechos.

Se puede demostrar para programas Datalog seguros que existe un punto a partir del cual no se pueden derivar más hechos verdaderos; es decir, para algún *k*, $I_{k+1} = I_k$. En este punto se tiene el conjunto final de hechos verdaderos. Además, dado un programa Datalog y la base de datos correspondiente, el procedimiento de punto fijo infiere todos los hechos verdaderos que se pueden inferir partiendo de dicha base de datos.

Si un programa recursivo contiene una regla con un literal negativo puede surgir un problema. Recuérdese que cuando se hizo una inferencia usando un ejemplar básico de una regla, por cada literal negativo **not** *q* en el cuerpo de la regla se comprobó que *q* no estaba presente en el conjunto de hechos *I*. Esta comprobación asume que *q* no se puede inferir después. Sin embargo, en la iteración del punto fijo, el conjunto de hechos *I* crece en cada iteración, e incluso si *q* no está presente en *I* en una iteración, puede aparecer más tarde. Así, podríamos haber hecho una inferencia en una iteración que no se pudo hacer en una iteración anterior, y la inferencia fue incorrecta. Se requiere que un programa recursivo no contenga literales negativos para evitar estos problemas.

En lugar de crear una vista para los empleados supervisados por Santos, se puede crear la vista *empl,* más general, que contenga todas las tuplas (*X*,*Y*) tales que *X* sea directa o indirectamente supervisado por *Y*, utilizando el siguiente programa (Figura 5.7):

empl (*X*, *Y*) :– *jefe* (*X*, *Y*)
empl (*X*, *Y*) :– *jefe* (*X*, *Z*), *empl* (*Z*, *Y*)

Para obtener los empleados supervisados directa o indirectamente por Santos, se utilizará la consulta

? *empl* (*X*, «Santos»)

que devuelve para *X* el mismo conjunto de valores de la vista *empl-Santos*. La mayoría de las implementaciones de Datalog cuentan con sofisticados optimizadores de consultas y motores de evaluación que pueden evaluar la consulta anterior a la misma velocidad que evaluarían la vista *empl-Santos*.

La vista *empl* definida anteriormente se denomina **cierre transitivo** de la relación *jefe*. Si se sustituyera la relación *jefe* por cualquier otra relación binaria *R*, el programa anterior definiría el cierre transitivo de *R*.

5.2.7. La potencia de la recursividad

Datalog con recursividad tiene mayor potencia expresiva que Datalog sin recursividad. En otras palabras, existen consultas en la base de datos que se pueden resol-

[1] La palabra «hecho» se usa en un sentido técnico para indicar la pertenencia de una tupla a una relación. Así, en el sentido de Datalog para «hecho», un hecho puede ser cierto (la tupla está realmente en la relación) o falso (la tupla no está en la relación).

ver utilizando recursividad, pero que no se pueden resolver sin utilizarla. Por ejemplo, en Datalog no se puede expresar el cierre transitivo sin utilizar recursividad (o, igualmente, en SQL o QBE sin recursividad). Considérese el cierre transitivo de la relación *jefe*. Intuitivamente, con un número fijo de reuniones pueden obtenerse solamente aquellos empleados que están ese número fijo de niveles por debajo de cualquier jefe (esta afirmación no se demostrará en este texto). Como cualquier consulta no recursiva tiene un número fijo de reuniones, existe un límite en el número de niveles de empleados que se pueden obtener mediante esa consulta. Si el número de niveles de empleados de la relación *jefe* es mayor que el límite de la consulta, no se encontrarán algunos niveles de empleados. Así, un programa Datalog no recursivo no puede expresar el cierre transitivo.

Una alternativa a la recursividad es utilizar un mecanismo externo, como SQL incorporado, para iterar una consulta no recursiva. La iteración implementa el bucle del algoritmo de punto fijo de la Figura 5.10. De hecho, así es como se implementan este tipo de consultas en los sistemas de bases de datos que no permiten la recursividad. Sin embargo, la formulación de estas consultas utilizando iteración es más complicada que utilizando recursividad y la evaluación utilizando recursividad puede optimizarse para que se ejecute más rápidamente que la evaluación utilizando iteración.

La potencia expresiva de la recursividad debe aprovecharse con cuidado. Es relativamente fácil escribir programas recursivos que generen un número infinito de hechos, como se muestra en el siguiente programa:

número (0)
número (A) :– *número* (B), $A = B + 1$

El programa genera *número* (n) para cualquier n positivo, es decir, para un conjunto infinito. La segunda regla del programa no cumple la condición de seguridad descrita en el Apartado 5.2.4. Los programas que cumplen la condición de seguridad terminarán, incluso si son recursivos, siempre que las relaciones de la base de datos sean finitas. Para programas de este tipo, las tuplas en las vistas pueden contener únicamente constantes de la base de datos, y por ello serán finitas. Lo opuesto no es cierto; es decir, existen programas que no satisfacen la condición de seguridad, pero que terminan.

5.2.8 Recursividad en otros lenguajes

La norma SQL:1999 soporta una forma limitada de recursión usando la cláusula **with recursive**. Supóngase que la relación *jefe* tiene los atributos *emp* y *mgr*. Podemos encontrar cada par (*X,Y*) tal que *X* tenga como jefe directo o indirecto a *Y* usando esta consulta SQL:1999:

```
with recursive empl (emp, jef) as (
        select emp, jef
        from jefe
    union
        select emp, empl.jef
        from jefe, empl
        where jefe.jef=empl.emp
    )
select *
from empl
```

Recuérdese que la cláusula **with** se usa para definir una vista temporal cuya definición sólo está disponible para la consulta en la que se define. La palabra clave adicional **recursive** especifica que la vista es recursiva. La definición SQL de la vista *empl* es equivalente a la versión Datalog que se vio en el Apartado 5.2.6.

El procedimiento PuntoFijo-Datalog utiliza iterativamente la función *inferir* (R, I) para obtener hechos verdaderos dado un programa Datalog recursivo. Aunque se considera únicamente el caso de los programas sin literales negativos, el procedimiento también se puede utilizar en las vistas definidas utilizando otros lenguajes, como SQL o el álgebra relacional, siempre que las vistas satisfagan las condiciones descritas a continuación. Independientemente del lenguaje que se utilice para definir una vista V, la vista se puede considerar definida por una expresión EV que, dado un conjunto de hechos I, devuelve un conjunto de hechos EV (I) para la relación V. Dado un conjunto de vistas R (definidas en cualquier lenguaje), se puede definir una función *inferir* (R, I) que devuelva $I \cup \bigcup_{V \in R} E_V (I)$. La función anterior es similar a la función *inferir* de Datalog.

Una vista V es **monótona** si, dado cualquier par de conjuntos de hechos I_1 e I_2, tales que $I_1 \subseteq I_2$, entonces $E_V (I_1) \subseteq E_V (I_2)$, donde E_V es la expresión utilizada para definir V. Análogamente, se dice que la función *inferir* es monótona si

$$I_1 \subseteq I_2 => \textit{inferir}\ (R, I_1) \subseteq \textit{inferir}\ (R, I_2)$$

Así, si *inferir* es monótona, dado un conjunto de hechos I_0 que es un subconjunto de hechos verdaderos, se puede asegurar que todos los hechos del conjunto *inferir* (R, I_0) son verdaderos. Utilizando el mismo razonamiento del Apartado 5.2.6, se puede demostrar que el procedimiento PuntoFijo-Datalog es correcto (es decir, calcula sólo los hechos verdaderos), ya que la función *inferir* es monótona.

Las expresiones del álgebra relacional que utilizan los operadores Π, σ, $\times$, $\bowtie$, $\cup$, $\cap$ o ρ son monótonas. Las vistas recursivas se pueden definir utilizando dichas expresiones.

Sin embargo, las expresiones relacionales que utilizan el operador – no son monótonas. Por ejemplo, sean $jefe_1$ y $jefe_2$ relaciones con el mismo esquema que la relación *jefe*. Sea

I_1 = { $jefe_1$ («Arteaga», «Benzoaga»), $jefe_1$ («Benzoaga», «Erejalde»), $jefe_2$ («Arteaga», «Benzoaga»)}

y sea

I_2 = { $jefe_1$ («Arteaga», «Benzoaga»), $jefe_1$ («Benzoaga», «Erejalde»), $jefe_2$ («Arteaga», «Benzoaga»), $jefe_2$ («Benzoaga», «Erejalde»)}

Considérese la expresión $jefe_1 - jefe_2$. El resultado de la expresión I_1 anterior es («Benzoaga»,«Erejalde»), mientras que el resultado de la expresión I_2 es la relación vacía. Se cumple que $I_1 \subseteq I_2$; según la definición, la expresión no es *monótona*. Las expresiones que se definen utilizando el operador de agrupación del álgebra relacional extendida tampoco son monótonas.

La técnica del punto fijo no funciona en vistas recursivas definidas con expresiones no monótonas. Sin embargo, existen situaciones en las que este tipo de vistas son útiles, en particular, para la definición de agregaciones en relaciones parte-subparte. Las relaciones de este tipo definen las subpartes que forman cada parte. Las subpartes pueden estar formadas a su vez por muchas subpartes y así sucesivamente; por tanto, las relaciones, como la relación jefe, tiene una estructura recursiva por naturaleza. Un ejemplo de una consulta de agregación sobre tal estructura sería calcular el número total de subpartes de cada parte. Escribir esta consulta en Datalog o en SQL (sin extensiones procedimentales) requeriría el uso de una vista recursiva sobre una expresión no monótona. Las notas bibliográficas contienen referencias sobre la investigación sobre la definición de vistas de este tipo.

Es posible definir algunos tipos de consultas recursivas sin utilizar vistas. Por ejemplo, se han propuesto operaciones relacionales extendidas para definir el cierre transitivo, y extensiones de la sintaxis SQL para especificar el cierre transitivo (generalizado). Sin embargo, las definiciones de vistas recursivas proporcionan una mayor potencia expresiva que las demás formas de consultas recursivas.

5.3. INTERFACES DE USUARIO Y HERRAMIENTAS

Aunque muchas personas interactúan con las bases de datos, pocas usan un lenguaje de consulta para interactuar directamente con un sistema de bases de datos. La mayoría interactúan mediante uno de los siguientes medios:

1. **Los formularios e interfaces gráficas de usuario** permiten a los usuarios introducir valores que completan las consultas predefinidas. El sistema ejecuta las consultas y da formato apropiado y muestra los resultados al usuario. Las interfaces gráficas de usuario proporcionan una forma fácil de interactuar con el sistema de bases de datos.
2. **Los generadores de informes** permiten generar informes predefinidos sobre los contenidos actuales de la base de datos. Los analistas o gestores examinan estos informes para tomar decisiones de negocio.
3. **Las herramientas de análisis de datos** permiten a los usuarios examinar interactivamente y analizar los datos.

Hay que hacer notar que estas interfaces usan lenguajes de consulta para comunicarse con los sistemas de bases de datos.

En este apartado se proporciona una visión general de los formularios, interfaces gráficas de usuario y generadores de informes. El Capítulo 22 cubre las herramientas de análisis de datos con más detalle. Por desgracia no hay normas para las interfaces de usuario, y cada sistema de base de datos proporciona usualmente su propia interfaz de usuario. En este apartado se describen los conceptos básicos, sin profundizar en los detalles de ningún producto en concreto.

5.3.1. Formularios e interfaces gráficas de usuario

Las interfaces de formularios se usan ampliamente para introducir y extraer datos en las bases de datos mediante consultas predefinidas. Por ejemplo, los motores World Wide Web proporcionan formularios que se usan para introducir palabras clave. Al pulsar el botón «Enviar» se provoca que el motor de búsqueda ejecute una consulta usando las palabras clave introducidas y que muestre el resultado al usuario.

Como otro ejemplo más orientado a las bases de datos, es posible conectarse a un sistema de matrícula de una universidad, donde se pide que se rellene un código y una contraseña en un formulario. El sistema usa esta información para comprobar la identidad, así como para extraer de la base de datos información, como el nombre y las asignaturas en las que el alumno se ha matriculado, y mostrarla. Puede haber más vínculos en la página Web que permitan buscar asignaturas y encontrar más información acerca de las asignaturas como el programa y el profesor.

Los exploradores Web compatibles con HTML constituyen los formularios e interfaces gráficas de usuario más ampliamente usados actualmente. La mayoría de

fabricantes de sistemas de bases de datos también proporcionan interfaces de formularios propietarias que ofrecen características más allá de las incluidas en los formularios HTML.

Los programadores pueden crear formularios e interfaces gráficas de usuario usando HTML o lenguajes de programación tales como C o Java. La mayoría de fabricantes de sistemas de bases de datos también proporcionan herramientas que simplifican la creación de formularios de una forma declarativa sencilla, usando programas de edición de formularios. Los usuarios pueden definir el tipo, tamaño y el formato de cada campo de un formulario usando el editor de formularios. Las acciones del sistema se pueden asociar con las acciones de los usuarios, como rellenar un campo, pulsar una tecla de función en el teclado o enviar un formulario. Por ejemplo, la ejecución de una consulta para rellenar los campos de nombre y dirección se pueden asociar con rellenar un campo de código, y la ejecución de una instrucción de actualización se puede asociar con el envío de un formulario.

Se pueden realizar comprobaciones de errores sencillas definiendo restricciones sobre los campos del formulario[2]. Por ejemplo, una restricción sobre el campo número de asignatura podría comprobar que el número de asignatura escrito por el usuario corresponde realmente con una asignatura. Aunque estas restricciones se pueden comprobar cuando se ejecuta la transacción, la detección temprana de errores ayuda a los usuarios a corregir rápidamente los errores. Los menús que indican los valores válidos que se pueden escribir en un campo pueden eliminar la posibilidad de muchos tipos de errores. Los desarrolladores de sistemas encuentran que su trabajo es mucho más fácil con la posibilidad de controlar tales características de forma declarativa la ayuda de una herramienta de desarrollo de interfaces de usuario, en lugar de crear el formulario directamente usando un lenguaje de guiones o de programación.

5.3.2. Generadores de informes

Los generadores de informes son herramientas que generan informes de resumen legibles de una base de datos. Integran la consulta a la base de datos con la creación de texto con formato y gráficos de resumen (tales como gráficos de barras o de tarta). Por ejemplo, un informe podría mostrar las ventas totales de los dos meses anteriores para cada zona de ventas.

El desarrollador de aplicaciones puede especificar formatos de informes usando las características de formato del generador de informes. Se pueden usar variables para almacenar parámetros tales como el mes y el año y para definir campos del informe. Se pueden definir tablas, gráficas de barras, gráficas de tarta u otros gráficos mediante consultas sobre la base de datos. Las definiciones de las consultas pueden usar valores de parámetros almacenados en las variables.

Una vez que se ha definido la estructura del informe con un generador de informes, se puede almacenar y ejecutar cada vez que se genere un informe. Los sistemas generadores de informes proporcionan varias características para estructurar una salida tabular, tal como definir las cabeceras de tabla y columna, mostrar subtotales por cada grupo en una tabla, dividir automáticamente una tabla grande en varias páginas y mostrar subtotales al final de cada página.

La Figura 5.13 es un ejemplo de un informe con formato. Los datos del informe se generan por agregación de la información sobre los pedidos.

La colección de herramientas de desarrollo de aplicaciones proporcionadas por los sistemas de bases de datos tales como los paquetes de formularios y los generadores de informes se conocen como *lenguajes de cuarta generación* (L4G). El nombre resalta que estas herramientas ofrecen un paradigma de la programación diferente del paradigma de programación imperativa ofrecido por los lenguajes de tercera generación como Pascal y C. Sin embargo, este término es menos relevante actualmente, dado que los formularios y los generadores de informes se crean normalmente con herramientas gráficas en lugar de con lenguajes de programación.

Compañía de Suministros Acme S. A.
Informe trimestral de ventas

Período: 1 de enero a 31 de marzo de 2001

Región	Categoría	Ventas	Subtotal
Norte	Hardware	1.000.000	
	Software	500.000	
	Todas las categorías		1.500.000
Sur	Hardware	200.000	
	Software	400.000	
	Todas las categorías		600.000

Ventas totales 2.100.000

FIGURA 5.13. Informe con formato.

[2] En Oracle se denominan «disparadores de formulario», pero en este libro se usará el término «disparador» con un sentido diferente que se estudia en el Capítulo 6.

RESUMEN

- Se han estudiado dos lenguajes de consulta: QBE y Datalog.
- QBE está basado en un paradigma visual: las consultas son muy similares a tablas.
- QBE y sus variantes se han hecho populares entre los usuarios poco expertos de base de datos, debido a la simplicidad intuitiva del paradigma visual. El sistema de bases de datos Access de Microsoft ampliamente usado soporta una versión gráfica de QBE denominada GQBE.
- Datalog se deriva de Prolog, pero a diferencia de éste, tiene una semántica declarativa que hace que las consultas sencillas sean más fáciles de formular y que la evaluación de consultas sea más fácil de optimizar.
- La definición de vistas es particularmente sencilla en Datalog y las vistas recursivas que permite Datalog hacen posible la formulación de consultas, tales como el cierre transitivo, que no podrían formularse sin utilizar recursividad o iteración. Sin embargo, en Datalog no existen normas para características importantes, como la agrupación y la agregación. Datalog sigue siendo principalmente un lenguaje de investigación.
- La mayoría de los usuarios interactúan con las bases de datos mediante formularios e interfaces gráficas de usuario, y hay muchas herramientas para simplificar la construcción de estas interfaces. Los generadores de informes son herramientas que ayudan a crear informes legibles a partir de los contenidos de la base de datos.

TÉRMINOS DE REPASO

- Caja de condición
- Cierre transitivo
- Cuadrícula de diseño
- Datalog
- Definición de vistas monótonas
- Depende
  - Directamente
  - Indirectamente
- Ejemplares
  - Ejemplar básico
  - Satisfacer
- Filas ejemplo
- Formularios
- Generadores de informes
- Graphical Query-by-Example (GQBE, consulta gráfica mediante ejemplos)
- Hecho
- Inferir
- Interfaces gráficas de usuario
- Literal negativo
- Literal positivo
- Microsoft Access
- Programa Datalog
- Punto fijo
- Query-by-Example (QBE, consulta mediante ejemplos)
- Regla
  - Cabeza
  - Cuerpo
- Reglas
- Relación resultado
- Seguridad
- Semántica
  - De una regla
  - De un programa
- Sintaxis en dos dimensiones
- Tablas esqueleto
- Vista no recursiva
- Vista recursiva

EJERCICIOS

5.1. Considérese la base de datos de seguros de la Figura 5.14, donde las claves primarias se identifican porque empiezan por letra mayúscula. Formúlense las siguientes consultas en QBE:

a. Buscar el número total de las personas cuyos coches se han visto involucrados en un accidente en 1989.

b. Buscar el número de accidentes en los cuales se ha visto involucrado un coche perteneciente a «Santos».

c. Añadir un nuevo accidente a la base de datos; supóngase cualquier valor para los atributos necesarios.

d. Borrar el Mazda de «Santos».

persona (*id-conductor*, *nombre*, *dirección*)
coche (*matrícula*, *año*, *modelo*)
accidente (*número-informe*, *fecha*, *lugar*)
es-dueño (*id-conductor*, *matrícula*)
participó (*id-conductor*, *coche*, *número-informe*, *importe-daños*)

FIGURA 5.14. Base de datos de seguros.

e. Actualizar el importe de daños del coche de matrícula «2002BCD» en el accidente con número de informe «AR2197» a 3.000 €.

5.2. Considérese la base de datos de empleados de la Figura 5.15. Proporciónense expresiones en QBE y Datalog para cada una de las consultas siguientes:

a. Buscar los nombres de todos los empleados que trabajan en el Banco Importante.
b. Buscar los nombres y ciudades de residencia de todos los empleados que trabajan en el Banco Importante.
c. Buscar los nombres, direcciones y ciudades de residencia de todos los empleados que trabajan en el Banco Importante y que ganan más de 10.000 €.
d. Buscar todos los empleados que viven en la ciudad de la empresa para la que trabajan.
e. Buscar todos los empleados que viven en la misma ciudad y en la misma calle que sus jefes.
f. Buscar todos los empleados que no trabajan en el Banco Importante.
g. Buscar todos los empleados que ganan más que cualquier empleado del Banco Pequeño.
h. Supóngase que las empresas pueden tener sede en varias ciudades. Buscar todas las empresas con sede en todas las ciudades en las que tiene sede el Banco Pequeño.

5.3. Considérese la base de datos relacional de la Figura 5.15, donde se han subrayado las claves primarias. Proporciónense expresiones en QBE y para cada una de las siguientes consultas:

a. Buscar todos los empleados que ganan más que el sueldo medio de los empleados de su empresa.
b. Buscar la empresa que tiene el mayor número de empleados.
c. Buscar la empresa que tiene el menor sueldo medio.
d. Buscar aquellas empresas cuyos empleados ganan un sueldo más alto, en media, que el sueldo medio del Banco Importante.

5.4. Considérese la base de datos relacional de la Figura 5.15. Proporciónense expresiones en QBE para cada una de las siguientes consultas:

a. Modificar la base de datos para que «Santos » viva en Tres Cantos.

empleado (*nombre-empleado*, *calle*, *ciudad*)
trabaja (*nombre-empleado*, *nombre-empresa*, *sueldo*)
empresa (*nombre-empresa*, *ciudad*)
jefe (*nombre-empleado*, *nombre-jefe*)

FIGURA 5.15. Base de datos de empleados.

b. Dar un 10% de aumento de suelto a todos los empleados del Banco Importante.
c. Dar un 10% de aumento de sueldo a todos los jefes de la base de datos.
d. Dar un 10% de aumento de sueldo a todos los jefes de la base de datos, a menos que su sueldo esté por encima de 100.000 € anuales. En tal caso, darles solamente un 3%.
e. Borrar todas las tuplas de la relación *trabaja* para los empleados del Banco Pequeño.

5.5. Sean los siguientes esquemas de relación:

$$R = (A, B, C)$$
$$S = (D, E, F)$$

Sean las relaciones $r(R)$ y $s(S)$. Proporciónense expresiones en QBE y Datalog equivalentes a cada una de las consultas siguientes:

a. $\Pi_A(r)$
b. $\sigma_{B=17}(r)$
c. $r \times s$
d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

5.6. Sea $R = (A, B, C)$ y sean r_1 y r_2 relaciones del esquema R. Proporciónense expresiones en QBE y Datalog equivalentes a cada una de las consultas siguientes:

a. $r_1 \cup r_2$
b. $r_1 \cap r_2$
c. $r_1 - r_2$
d. $\Pi_{AB}(r_1) \bowtie \Pi_{BC}(r_2)$

5.7. Sean $R = (A, B, C)$ y $S = (A, C)$, y sean $r(R)$ y $s(S)$ relaciones. Escríbanse expresiones en QBE y Datalog para cada una de las siguientes consultas:

a. $\{<a> \mid \exists\, b\,(<a, b> \in r \land b = 17)\}$
b. $\{<a, b, c> \mid <a, b> \in r \land <a, c> \in s\}$
c. $\{<a> \mid \exists\, c\,(<a, c> \in s \land \exists\, b_1, b_2\,(<a, b_1> \in r \land <c, b_2> \in r \land b_1 > b_2))\}$

5.8. Considérese la base de datos relacional de la Figura 5.12. Escríbase un programa Datalog para cada una de las siguientes consultas:

a. Buscar todos los empleados que trabajan (directa o indirectamente) bajo la dirección de «Santos».
b. Buscar las ciudades de residencia de todos los empleados que trabajan (directa o indirectamente) bajo la dirección de «Santos».
c. Buscar todas las parejas de empleados que tienen un jefe (directo o indirecto) en común.
d. Buscar todas las parejas de empleados que tienen un jefe (directo o indirecto) en común y que están el mismo número de niveles bajo el jefe.

5.9. Escríbase una vista del álgebra relacional extendida equivalente a la regla Datalog

$$p(A, C, D) \;:-\; q1\,(A, B),\, q2\,(B, C),\, q3\,(4, B),\, D = B + 1$$

5.10. Descríbase cómo una regla de Datalog arbitraria puede expresarse como una vista del álgebra relacional extendida.

NOTAS BIBLIOGRÁFICAS

La versión experimental de Query-by-Example se describe en Zloof [1977]; la versión comercial se describe en IBM [1978b]. Muchos sistemas de bases de datos —en concreto, sistemas de bases de datos que se ejecutan en computadoras personales— implementan QBE o variantes. Access de Microsoft y Paradox de Borland son ejemplos de ello.

Algunas implementaciones de Datalog son el sistema LDL (descrito en Tsur y Zaniolo [1986] y Navqi y Tsur [1988]), Nail! (descrito en Derr et al. [1993]) y Coral (descrito en Ramakrishnan et al. [1992b] y en Ramakrishnan et al. [1993]). Las primeras discusiones sobre bases de datos lógicas se presentaron en Gallaire y Minker [1978] y en Gallaire et al. [1984]. Ullman [1988] y Ullman [1989] proporciona discusiones extensas de lenguajes lógicos de consulta y de técnicas de implementación. Ramakrishnan y Ullman [1995] proporcionan una visión de conjunto más reciente sobre bases de datos deductivas.

A los programas Datalog que tienen tanto recursividad como negación se les puede asignar una semántica sencilla si se «estratifica» la negación, es decir, si no hay recursividad durante la negación. La negación estratificada se discute en Chandra y Harel [1982] y en Apt y Pugin [1987]. Una extensión importante, llamada *semántica de estratificación modular*, que maneja una clase de programas recursivos con literales negativos, se discute en Ross [1990]; en Ramakrishnan et al. [1992c] se describe una técnica de evaluación para tales programas.

HERRAMIENTAS

QBE de Access de Microsoft es probablemente la implementación de QBE más ampliamente usada. QMF de DB2 de IBM y Paradox de Borland también incluyen QBE.

El sistema Coral de la Universidad de Wisconsin - Madison es una implementación de Datalog ampliamente usada (véase http://www.cs.wisc.edu/coral). El sistema XSB de la Universidad estatal de Nueva York (SUNY) Stony Brook (http://xsb.sourceforge.net) es una implementación de Prolog ampliamente usada que soporta consultas a bases de datos; recuérdese que Datalog es un subconjunto no procedimental de Prolog.

CAPÍTULO

6 INTEGRIDAD Y SEGURIDAD

Las restricciones de integridad proporcionan un medio de asegurar que las modificaciones hechas a la base de datos por los usuarios autorizados no provoquen la pérdida de la consistencia de los datos. Por tanto, las restricciones de integridad protegen a la base de datos contra los daños accidentales.

En el Capítulo 2 ya se ha visto una modalidad de restricciones de integridad para el modelo E-R. Estas restricciones eran de los tipos siguientes:

- **Declaración de claves** – la estipulación de que ciertos atributos pueden formar una clave para un conjunto de entidades determinado.
- **Forma de la relación** – de varios a varios, de uno a varios, de uno a uno.

En general, la restricción de integridad puede ser un predicado arbitrario referente a la base de datos. Sin embargo, los predicados arbitrarios pueden resultar complicados de verificar. En consecuencia, lo habitual es limitarse a restricciones de integridad que puedan verificarse con una sobrecarga mínima. En los apartados 6.1 y 6.2 se estudian estas formas de restricciones de integridad y una forma más compleja en el Apartado 6.3. En el Capítulo 7 se estudia otra forma de restricción de integridad, denominada «dependencia funcional», que se usa principalmente en el proceso del diseño de esquemas.

En el Apartado 6.4 se estudian los *disparadores*, que son instrucciones que el sistema ejecuta automáticamente como efecto colateral de una modificación de la base de datos. Los disparadores se usan para asegurar algunos tipos de integridad.

Además de la protección contra la introducción accidental de inconsistencia, puede ser necesario proteger los datos almacenados en la base de datos frente a accesos no autorizados y destrucción o alteración malintencionada. En los apartados 6.5 hasta el 6.7 se examinan formas en que se puede hacer un mal uso de los datos o hacerlos intencionadamente inconsistentes, y se presentan mecanismos de seguridad para protegerse contra ello.

6.1. RESTRICCIONES DE LOS DOMINIOS

Se ha visto que hay que asociar a cada atributo un dominio de valores posibles. En el Capítulo 4 se vieron varios tipos de dominios estándar, tales como los enteros, caracteres y fecha/tiempo en SQL. La declaración de que un atributo pertenezca a un determinado dominio actúa como una restricción sobre los valores que puede tomar. Las restricciones de los dominios son la forma más simple de restricción de integridad. El sistema las verifica fácilmente siempre que se introduce en la base de datos un nuevo elemento de datos.

Es posible que varios atributos tengan el mismo dominio. Por ejemplo, los atributos *nombre-cliente* y *nombre-empleado* pueden tener el mismo dominio: el conjunto de los nombres de persona. Sin embargo, los dominios de *saldo* y de *nombre de la sucursal* deben ser, ciertamente, diferentes. Quizá resulte menos evidente si *nombre-cliente* y *nombre-sucursal* deben tener el mismo dominio. En el nivel de implementación, tanto los nombres de los clientes como los de las sucursales son cadenas de caracteres. Sin embargo, normalmente no se considerará que la consulta «Hallar todos los clientes que tengan el nombre de una sucursal» tenga sentido. Por tanto, si se considera la base de datos desde el punto de vista teórico, en vez de hacerlo desde el punto de vista físico, *nombre-cliente* y *nombre-sucursal* deben tener dominios diferentes.

De la discusión anterior se puede deducir que una definición adecuada de las restricciones de los dominios no sólo permite verificar los valores introducidos en la base de datos, sino también examinar las consultas para asegurarse de que tengan sentido las comparaciones que hagan. El principio subyacente a los dominios de los atributos es parecido al de los tipos de las variables en los lenguajes de programación. Los lenguajes de programación con tipos estrictos permiten al compilador examinar el programa con mayor detalle.

La cláusula **create domain** se puede usar para definir nuevos dominios. Por ejemplo, las instrucciones:

create domain *Euros* **numeric**(12,2)
create domain *Dólares* **numeric**(12,2)

definen los dominios *Euros* y *Dólares* como números decimales con un total de 12 dígitos, dos de los cuales se sitúan después de la coma decimal. Un intento de asignar un valor de tipo *Dólares* a una variable de tipo *Euros* resultaría en un error sintáctico, aunque ambos tengan el mismo tipo numérico. Tal asignación probablemente es debida a un error del programador, en el que este olvidó las diferencias de cambio. La declaración de diferentes dominios para diferentes monedas ayuda a detectar errores.

Los valores de un dominio pueden ser *convertidos* a otro dominio. Si el atributo *A* de la relación *r* es de tipo *Euros*, se puede convertir a *Dólares* escribiendo:

cast *r.A* **as** *Dólares*

En una aplicación real se multiplicaría *r.A* por el factor de cambio antes de convertirlo a dólares. SQL también proporciona las cláusulas **drop domain** y **alter domain** para borrar o modificar dominios que se hayan declarado anteriormente.

La cláusula **check** de SQL permite restringir los dominios de maneras poderosas que no permiten la mayor parte de los sistemas de tipos de los lenguajes de programación. Concretamente, la cláusula **check** permite al diseñador del esquema especificar un predicado que debe satisfacer cualquier valor asignado a una variable cuyo tipo sea el dominio. Por ejemplo, una cláusula **check** puede asegurar que un dominio de sueldo por hora sólo permita valores mayores que un valor especificado (como puede ser el sueldo mínimo), tal y como se muestra aquí:

create domain *sueldo-por-hora* **numeric**(5,2)
    constraint *comprobación-valor-sueldo*
        check(**value** ≥ 4.00)

El dominio *sueldo-por-hora* tiene una restricción que asegura que el sueldo por hora sea mayor que 4,00. La orden **constraint** *comprobación-valor-sueldo* es opcional y se utiliza para dar a la restricción el nombre de *comprobación-valor-sueldo*. El nombre se utiliza para indicar la restricción violada por una actualización.

La cláusula **check** también puede utilizarse para restringir un dominio para que no contenga valores nulos, como se muestra aquí:

create domain *número-cuenta* **char**(10)
    constraint *comprobación-número—cuenta-nulo*
        check(**value not null**)

En este otro ejemplo el dominio se puede limitar para que contenga sólo un conjunto especificado de valores usando la cláusula **in**:

create domain *tipo-cuenta* **char**(10)
    constraint *comprobación-tipo-cuenta*
        check(**value in** ('Corriente', 'Ahorro'))

Las condiciones **check** anteriores se puede comprobar muy fácilmente cuando se inserta o modifica una tupla. Sin embargo, en general, las condiciones **check** pueden ser muy complejas (y difíciles de comprobar) dado que se permiten subconsultas que se refieren a otras relaciones en la condición **check**. Por ejemplo, esta restricción se podría especificar sobre la relación *préstamo*:

check (nombre-sucursal **in**
    (**select** nombre-sucursal **from** sucursal))

La condición **check** verifica que *nombre-sucursal* en cada tupla en la relación *préstamo* es realmente el nombre de una sucursal de la relación *cuenta*. Así, la condición no sólo se debe comprobar cuando se inserte o modifique *préstamo*, sino también cuando cambie la relación *sucursal* (en este caso, cuando se borre o modifique *cuenta*).

La restricción anterior es realmente un ejemplo de una clase de restricciones denominadas restricciones de *integridad referencial*. En el Apartado 6.2 se estudian estas restricciones y la forma de especificarlas de forma más sencilla en SQL.

Las condiciones **check** complejas pueden ser útiles cuando se desee asegurar la integridad de los datos, pero se deben usar con cuidado, dado que pueden ser costosas de comprobar.

6.2. INTEGRIDAD REFERENCIAL

A menudo se desea asegurar que un valor que aparece en una relación para un conjunto de atributos determinado aparezca también en otra relación para un cierto conjunto de atributos. Esta condición se denomina **integridad referencial**.

6.2.1. Conceptos básicos

Considérese un par de relaciones r (R) y s (S) y la reunión natural $r \bowtie s$. Puede haber una tupla t_r de r que no se reúna con ninguna tupla de s. Es decir, no hay ningún t_s en s

tal que $t_r [R \cap S] = t_s [R \cap S]$. Estas tuplas se denominan *colgantes*. Las tuplas colgantes pueden ser aceptables en función del conjunto de entidades o de relaciones que se esté modelando. En el Apartado 3.5.2 se tomó en consideración una forma de reunión modificada —la reunión externa— para operar con las relaciones que contengan tuplas colgantes. En este caso el motivo de preocupación no son las consultas, sino el momento en que se desea permitir que haya tuplas colgantes en la base de datos.

Supóngase que hay una tupla t_1 en la relación *cuenta* con t_1[*nombre-sucursal*] = «As Pontes» pero que no hay ninguna tupla de la relación *sucursal* para la sucursal de As Pontes. Esta situación no es deseable. Se espera que la relación *sucursal* muestre una relación de todas las sucursales bancarias. Por tanto, la tupla t_1 hará referencia a una cuenta de una sucursal que no existe. Evidentemente, es preferible tener una restricción de integridad que prohíba las tuplas colgantes de este tipo.

Sin embargo, algunos casos de tuplas colgantes pueden resultar convenientes. Supóngase que hay una tupla t_2 de la relación *sucursal* con t_2[*nombre-sucursal*] = «Gijón» pero que no hay ninguna tupla de la relación *cuenta* para la sucursal de Gijón. En este caso hay una sucursal que no tiene ninguna cuenta. Aunque esta situación no es frecuente, puede producirse cuando se abre una sucursal o cuando está a punto de cerrar. Por tanto, no es deseable prohibir esta situación.

La distinción entre estos dos ejemplos se basa en dos hechos:

- El atributo *nombre-sucursal* de *Esquema-cuenta* es una clave externa que hace referencia a la clave primaria de *Esquema-sucursal*.
- El atributo *nombre-sucursal* de *Esquema-sucursal* no es una clave externa.

(Recuérdese del Apartado 3.1.3 que una clave externa es un conjunto de atributos del esquema de una relación que forma la clave primaria de otro esquema.)

En el ejemplo de As Pontes la tupla t_1 de *cuenta* tiene un valor en la clave externa *nombre-sucursal* que no aparece en *sucursal*. En el ejemplo de la sucursal de Gijón, la tupla t_2 de *sucursal* tiene un valor de *nombre-sucursal* que no aparece en *cuenta*, pero *nombre-sucursal* no es una clave externa. Por tanto, la distinción entre los dos ejemplos de tuplas colgantes es la presencia de una clave externa.

Sean $r_1(R_1)$ y $r_2(R_2)$ dos relaciones con las claves primarias K_1 y K_2, respectivamente. Se dice que un subconjunto α de R_2 es una **clave externa** que hace referencia a K_1 de la relación r_1 si se exige que para cada t_2 de r_2 haya una tupla t_1 de r_1 tal que $t_1[K_1] = t_2[\alpha]$. Las exigencias de este tipo se denominan **restricciones de integridad referencial** o **dependencia de subconjunto**. El último término se debe a que esta última restricción de integridad referencial puede escribirse como $\Pi_\alpha (r_2) \subseteq \Pi_{K_1} (r_1)$. Obsérvese que, para que una restricción de integridad referencial tenga sentido, α debe ser igual a K_1, o bien α y K_1 deben ser conjuntos compatibles de atributos.

6.2.2. Integridad referencial en el modelo E-R

Las restricciones de integridad referencial aparecen con frecuencia. Si se obtiene el esquema de la base de datos relacional creando tablas a partir de los diagramas E-R, tal y como se vio en el Capítulo 2, cada relación que proceda de un conjunto de relaciones tendrá restricciones de integridad referencial. En la Figura 6.1 se muestra un conjunto *n*-ario de relaciones R, que relaciona los conjuntos de entidades $E_1, E_2, \ldots, E_n$. K_i denota la clave primaria de E_i. Los atributos del esquema de la relación del conjunto de relaciones R incluyen $K_1 \cup K_2 \cup \ldots \cup K_n$. Cada K_i del esquema de R es una clave externa que lleva a una restricción de integridad referencial.

Otra fuente de restricciones de integridad referencial son los conjuntos de entidades débiles. Recuérdese del Capítulo 2 que el esquema de la relación de un conjunto de entidades débiles debe incluir la clave primaria del conjunto de entidades del que este depende. Por tanto, el esquema de la relación de cada conjunto de entidades débiles incluye una clave externa que lleva a una restricción de integridad referencial.

6.2.3. Modificación de la base de datos

La modificación de la base de datos puede ocasionar violaciones de la integridad referencial. A continuación se describe la comprobación que debe hacerse para cada tipo de modificación de la base de datos para preservar la siguiente restricción de integridad referencial:

$$\Pi_\alpha (r_2) \subseteq \Pi_K (r_1)$$

- **Insertar.** Si se inserta una tupla t_2 en r_2, el sistema debe asegurar que hay una tupla t_1 de r_1 tal que $t_1[K] = t_2[\alpha]$. Es decir,

$$t_2[\alpha] \in \Pi_K (r_1)$$

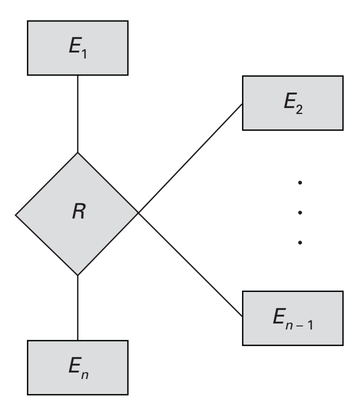


FIGURA 6.1. Un conjunto de relaciones N-ario.

- **Borrar.** Si se borra una tupla t_1 de r_1 el sistema debe calcular el conjunto de tuplas de r_2 que hacen referencia a r_1:

$$\sigma_{\alpha = t_1[K]}(r_2)$$

Si este conjunto no es el conjunto vacío, o bien se rechaza la orden borrar como error, o bien se deben borrar las tuplas que hacen referencia a t_1. La última solución puede llevar a borrados en cascada, dado que las tuplas pueden hacer referencia a tuplas que hagan referencia a t_1, etcétera.

- **Actualizar.** Hay que considerar dos casos: las actualizaciones de la relación que realiza la referencia (r_2) y las actualizaciones de la relación a la que se hace referencia (r_1).
  - — Si se actualiza la tupla t_2 de la relación r_2 y esta actualización modifica valores de la clave externa α, se realiza una comprobación parecida a la del caso de la inserción. Si t'_2 denota el nuevo valor de la tupla t_2, el sistema debe asegurar que

$$t'_2[\alpha] \in \Pi_K(r_1)$$

  - — Si se actualiza la tupla t_1 de la relación r_1 y esta actualización modifica valores de la clave primaria (K), se realiza una comprobación parecida a la del caso del borrado. El sistema debe asegurar que

$$\sigma_{\alpha = t_1[K]}(r_2)$$

utilizando el valor anterior de t_1 (el valor antes de que se lleve a cabo la actualización). Si este conjunto no es el conjunto vacío, la actualización se rechaza como error o se ejecuta en cascada de manera parecida al borrado.

6.2.4. Integridad referencial en SQL

Las claves primarias pueden especificarse como parte de la instrucción **create table** de SQL usando la cláusula **foreign key**. Se ilustrarán las declaraciones de clave externa usando la definición del LDD de SQL de parte de la base de datos bancaria, mostrada en la Figura 6.2.

De manera predeterminada, una clave externa referencia los atributos que forman la clave primaria de la tabla referenciada. SQL también soporta una versión de la cláusula **references**, donde se puede especificar explícitamente una lista de atributos de la relación referenciada. Esta lista se debe declarar como clave candidata de la relación referenciada.

Se puede usar la siguiente forma abreviada como parte de la definición de un atributo para declarar que el atributo forma una clave externa:

nombre-sucursal **char**(15) **references** *sucursal*

Cuando se viola una restricción de integridad referencial, el procedimiento normal es rechazar la acción

```
create table cliente
    (nombre-cliente     char(20),
    calle-cliente       char(30),
    ciudad-cliente      char(30),
    primary key (nombre-cliente))

create table sucursal
    (nombre-sucursal    char(15),
    ciudad-sucursal     char(30),
    activo              integer,
    primary key (nombre-sucursal),
    check (activo > = 0))

create table cuenta
    (número-cuenta      char(10),
    nombre-sucursal     char(15),
    saldo               integer,
    primary key (número-cuenta),
    foreign key (nombre-sucursal) references sucursal,
    check (saldo >= 0))

create table impositor
    (nombre-cliente     char(20),
    número-cuenta       char(10),
    primary key (nombre-cliente, número-cuenta),
    foreign key (nombre-cliente) references cliente,
    foreign key (número-cuenta) references cuenta)
```

FIGURA 6.2. Definición en SQL de los datos de parte de la base de datos bancaria.

que provocó la violación. Sin embargo, la cláusula **foreign key** puede especificar que si una acción de borrado o de actualización de la relación a la que hace referencia viola la restricción, en lugar de rechazar la acción, hay que adoptar medidas para modificar la tupla de la relación que hace la referencia con objeto de restaurar la restricción. Considérese la siguiente definición de una restricción de integridad de la relación *cuenta*:

```
create table cuenta
    (…
    foreign key (nombre-sucursal) references sucursal
                                  on delete cascade
                                  on update cascade,
    …)
```

Debido a la cláusula **on delete cascade** asociada con la declaración de la clave externa, si un borrado de una tupla de *sucursal* da lugar a que se viole la restricción de integridad referencial, el sistema no rechaza el borrado. En su lugar, el borrado se realiza en «cascada» en la relación *cuenta*, borrando la tupla que hace referencia a la sucursal que se borró. De modo parecido, no se rechaza la actualización de un campo al que haga referencia la restricción si viola esta; en vez de eso, el campo *nombre-sucursal* de las tuplas que realizan la referencia de *cuenta* se actualizan también al nuevo valor. SQL también permite que la cláusula **foreign key** especifique una acción diferente a **cascade** si se viola la restricción: el campo que hace la referencia (en este caso, *nombre-*

sucursal) se puede establecer en nulo o darle un valor predeterminado para el dominio (usando **set default**).

Si hay una cadena de dependencias de claves externas entre varias relaciones, un borrado o una actualización en uno de sus extremos puede propagarse por toda la cadena. En el Ejercicio 6.4 se toma en consideración un caso interesante en el que la restricción **foreign key** de una relación hace referencia a esa misma relación. Si una actualización o borrado en cascada crea una violación de la restricción que no puede resolverse mediante una nueva operación en cascada, el sistema aborta la transacción. En consecuencia, todas las modificaciones generadas por la transacción y por sus acciones en cascada se deshacen.

En SQL la semántica de las claves se complica por el hecho de que SQL permite valores **nulos**. Los atributos de las claves externas pueden ser nulos, dado que no se han declarado como no nulos. Si todas las columnas de una clave externa contienen nulos para una tupla en concreto, se usa para esta tupla la definición usual de las restricciones de clave externa. Si alguna de las columnas contiene un valor nulo, la tupla se define automáticamente como que cumple la restricción.

Puede que esta definición no sea siempre la mejor elección, así que SQL proporciona constructores que permiten cambiar el comportamiento con los valores nulos; aquí no se tratan estos constructores. Para evitar esta complejidad, es mejor asegurarse de que todas las columnas con especificaciones **foreign key** se declaren como no nulas.

Las transacciones pueden consistir en varios pasos, y las restricciones de integridad se pueden violar temporalmente después de un paso, pero en un paso posterior la violación puede desaparecer. Por ejemplo, supóngase que se tiene la relación *personacasada* con clave primaria *nombre* y un atributo *cónyuge*, y supónganse que *cónyuge* es una clave externa de *personacasada*. Es decir, la restricción dice que el atributo *cónyuge* debe contener un nombre que aparezca en la tabla *persona*. Supóngase que se desea añadir el hecho de que Juan y María están casados insertando dos tuplas, una para Juan y otra para María en la relación anterior. La inserción de la primera tupla violaría la restricción de clave externa, independientemente de cuál de las dos tuplas se haya insertado primero. Después de que se inserte la segunda tabla, la restricción de clave externa se volvería a cumplir.

Para manejar estas situaciones las restricciones de integridad se comprueban al final de la transacción en lugar de en los pasos intermedios[1].

6.3. ASERTOS

Un **aserto** es un predicado que expresa una condición que se desea que la base de datos satisfaga siempre. Las restricciones de dominio y las de integridad referencial son formas especiales de los asertos. Se ha prestado una atención especial a estos tipos de asertos porque se pueden verificar con facilidad y se aplican a una gran variedad de aplicaciones de bases de datos. Sin embargo, hay muchas restricciones que no se pueden expresar utilizando únicamente estas formas especiales. Ejemplos de estas restricciones pueden ser

- La suma de todos los importes de los préstamos de cada sucursal debe ser menor que la suma de todos los saldos de las cuentas de esa sucursal.
- Cada préstamo tiene al menos un cliente que tiene una cuenta con un saldo mínimo de 1.000 €.

En SQL los asertos adoptan la forma

create assertion <nombre-aserto> **check** <predicado>

Las dos restricciones mencionadas pueden escribirse tal y como se muestra a continuación. Dado que SQL no proporciona ningún mecanismo «para todo *X*, *P*(*X*)» (donde *P* es un predicado), no queda más remedio que implementarlo utilizando su equivalente «no existe *X* tal que no *P*(*X*)», que puede escribirse en SQL.

```
create assertion restricción-suma check
    (not exists (select * from sucursal
        where (select sum(importe) from préstamo
            where préstamo.nombre-sucursal
                = sucursal.nombre-sucursal)
          >= (select sum (importe) from cuenta
            where préstamo.nombre-sucursal
                = sucursal.nombre-sucursal)))

create assertion restricción-saldo check
    (not exists (select * from préstamo
        where not exists (select *
            from prestatario, impositor, cuenta
            where préstamo.número-préstamo
                = prestatario.número-préstamo
              and prestatario.nombre-prestatario
                = impositor.nombre-cliente
              and impositor.número-cuenta
                = cuenta.número-cuenta
              and cuenta.saldo >= 1000)))
```

[1] El problema de este ejemplo se puede resolver de otra forma si el atributo *cónyuge* puede ser nulo: se ponen los atributos *cónyuge* a nulo al insertar las tuplas de Juan y María, y se actualizan más tarde. Sin embargo, esta técnica no es aconsejable y no funciona si los atributos no pueden tomar el valor nulo.

Cuando se crea un aserto el sistema comprueba su validez. Si el aserto es válido, sólo se permiten las modificaciones posteriores de la base de datos que no hagan que se viole el aserto. Esta comprobación puede introducir una sobrecarga importante si se han realizado asertos complejos. Por tanto, los asertos deben utilizarse con mucha cautela. La elevada sobrecarga debida a la comprobación y al mantenimiento de los asertos ha llevado a algunos desarrolladores de sistemas a soslayar el soporte para los asertos generales, o bien a proporcionar formas especializadas de aserto que resultan más sencillas de verificar.

6.4. DISPARADORES

Un *disparador* es una orden que el sistema ejecuta de manera automática como efecto secundario de la modificación de la base de datos. Para diseñar un mecanismo disparador hay que cumplir dos requisitos:

1. Especificar las condiciones en las que se va a ejecutar el disparador. Esto se descompone en un *evento* que causa la comprobación del disparador y una *condición* que se debe cumplir para ejecutar el disparador.
2. Especificar las *acciones* que se van a realizar cuando se ejecute el disparador.

Este modelo de disparadores se denomina modelo **evento-condición-acción**.

La base de datos almacena disparadores como si fuesen datos normales, por lo que son persistentes y accesibles para todas las operaciones de la base de datos. Una vez se almacena un disparador en la base de datos, el sistema de base de datos asume la responsabilidad de ejecutarlo cada vez que ocurra el evento especificado y se satisfaga la condición correspondiente.

6.4.1. Necesidad de los disparadores

Los disparadores son mecanismos útiles para alertar a los usuarios o para realizar de manera automática ciertas tareas cuando se cumplen determinadas condiciones. A modo de ejemplo supóngase que, en lugar de permitir saldos de cuenta negativos, el banco trata los descubiertos dejando a cero el saldo de las cuentas y creando un préstamo por el importe del descubierto. Este préstamo recibe un número de préstamo idéntico al número de cuenta que ha tenido el descubierto. En este ejemplo la condición para ejecutar el disparador es una actualización de la relación *cuenta* que dé lugar a un valor negativo de *saldo*. Supóngase que Santos retiró cierta cantidad de dinero de una cuenta que dio lugar a que el saldo de la cuenta fuera negativo. *t* denota la tupla de la cuenta con un valor negativo de *saldo*. Las acciones que hay que emprender son las siguientes:

- Insertar una nueva tupla *s* a la relación *préstamo* con

  s[*nombre-sucursal*] = *t*[*nombre-sucursal*]
  s[*número-préstamo*] = *t*[*número-cuenta*]
  s[*importe*] = – *t*[*saldo*]

  (Obsérvese que, dado que *t*[*saldo*] es negativo, hay que cambiar el signo de *t*[*saldo*] para obtener el importe del préstamo – un número positivo).

- Insertar una nueva tupla *u* a la relación *prestatario* con

  u[*nombre-cliente*] = «Santos»
  u[*número-préstamo*] = *t*[*número-cuenta*]

- Hacer que *t*[*saldo*] sea 0.

Como otro ejemplo del uso de disparadores, supóngase un almacén que desee mantener un inventario mínimo de cada producto; cuando el nivel de inventario de un producto cae por debajo del nivel mínimo, se debería solicitar un pedido automáticamente. Para implementar con disparadores esta regla de negocio se haría: al modificar el nivel de inventario de un producto, el disparador debería comparar el nivel con el mínimo y si el nivel es inferior al mínimo, se añadiría un nuevo pedido a la relación *pedido*.

Obsérvese que los sistemas de disparadores en general no pueden realizar actualizaciones fuera de la base de datos, y por lo tanto, en este último ejemplo, no se puede usar un disparador para realizar un pedido en el mundo externo. En su lugar se añade un pedido a la relación *pedidos* como en el ejemplo del inventario. Se debe crear un proceso del sistema separado que se esté ejecutando constantemente que explore periódicamente la relación *pedidos* y solicite los pedidos. Este proceso del sistema anotaría las tuplas de la relación *pedidos* que se han procesado y cuándo se ha procesado cada pedido. El proceso también llevaría cuenta del despacho de pedidos y alertaría a los gestores en caso de condiciones excepcionales tales como retrasos en las entregas.

6.4.2. Disparadores en SQL

Los sistemas de bases de datos SQL usan ampliamente los disparadores, aunque antes de SQL:1999 no fueron parte de la norma. Por desgracia, cada sistema de bases de datos implementó su propia sintaxis para los disparadores, conduciendo a incompatibilidades. En la Figura 6.3 se describe la sintaxis SQL:1999 para los disparadores (que es similar a la sintaxis de los sistemas de bases de datos DB2 de IBM y de Oracle).

```
create trigger descubierto after update on cuenta
referencing new row as nfila
for each row
when nfila.saldo < 0
begin atomic
    insert into prestatario
        (select nombre-cliente, número-cuenta
        from impositor
        where nfila.número-cuenta = impositor.número-cuenta);
    insert into préstamo values
        (nfila.número-cuenta, nfila.nombre-sucursal, – nfila.saldo)
    update cuenta set saldo = 0
        where cuenta.número-cuenta = nfila.número-cuenta
end
```

FIGURA 6.3. Ejemplo de la sintaxis de SQL:1999 para los disparadores.

Esta definición de disparador especifica que el disparador se inicia *después* (**after**) de cualquier actualización de la relación *cuenta*. Una instrucción de actualización SQL podría actualizar múltiples tuplas de la relación, y la cláusula **for each row** en el código del disparador se iteraría explícitamente por cada fila actualizada. La cláusula **referencing new row as** crea una variable *nfila* (denominada **variable de transición**) que almacena el valor de una fila actualizada después de la actualización.

La instrucción **when** especifica una condición, en este caso *nfila.saldo* < 0. El sistema ejecuta el resto del cuerpo del disparador sólo para las tuplas que satisfacen la condición. La cláusula **begin atomic** … **end** sirve para encuadrar varias instrucciones SQL en una única instrucción compuesta. Las dos instrucciones **insert** en la estructura **begin** … **end** realizan las tareas específicas para la creación de nuevas tuplas en las relaciones *prestatario* y *préstamo* para representar el nuevo préstamo. La instrucción **update** sirve para establecer en 0 el saldo de la cuenta.

El evento del disparador puede tener varias formas:

- El *evento* del disparador puede ser **insert** o **delete** en lugar de **update**.

  Por ejemplo, la acción sobre el borrado (**delete**) de una cuenta podría comprobar si los tenedores de la cuenta tienen otras cuentas y, si no las tienen, borrarlas de la relación *impositor*. Se deja al lector la definición de este disparador como ejercicio (Ejercicio 6.7).

  Como otro ejemplo, si se inserta un nuevo *impositor*, la acción del disparador podría ser enviar una carta de bienvenida al impositor. Obviamente, un disparador no puede causar directamente esta acción fuera de la base de datos, pero en su lugar sí puede insertar una nueva tupla a una relación que almacene las direcciones a las que se deben enviar las cartas de bienvenida. Un proceso separado examinaría esta relación e imprimiría las cartas a enviar.

- Para las actualizaciones el disparador puede especificar columnas cuya actualización cause la ejecución del disparador. Por ejemplo, si la primera línea del disparador de descubiertos se reemplazase por

  create trigger *descubierto* **after update of** *saldo* **on** *cuenta*

  entonces el disparador se ejecutaría sólo cuando se actualizase *saldo*; las actualizaciones del resto de atributos no causarían su ejecución.

- La cláusula **referencing old row as** se puede usar para crear una variable que almacene el valor anterior de una fila actualizada o borrada. La cláusula **referencing new row as** se puede usar con las inserciones además de las actualizaciones.

- Los disparadores se pueden activar antes (**before**) del evento (**insert/delete/update**) en lugar de después (**after**) del evento.

  Estos disparadores pueden servir como restricciones extras que pueden evitar actualizaciones no válidas. Por ejemplo, si no se desea permitir descubiertos, se puede crear un disparador **before** que retroceda la transacción si el nuevo saldo es negativo.

  Como otro ejemplo, supóngase que el valor del campo de número telefónico de una tupla insertada está vacío, que indica la ausencia de un número de teléfono. Se puede definir un disparador que reemplace el valor por el valor **null**. Se puede usar la instrucción **set** para realizar estas modificaciones.

  ```
  create trigger poner-nulo before update on r
  referencing new row as nfila
  for each row
  when nfila.número-teléfono = ' '
  set nfila.número-teléfono = null
  ```

- En lugar de realizar una acción por cada fila afectada, se puede realizar una acción única para la instrucción SQL completa que causó la inserción, borrado o actualización. Para hacerlo se usa la cláusula **for each statement** en lugar de **for each row**.

  Las cláusulas **referencing old table as** o **referencing new table as** se pueden usar entonces para

hacer referencia a tablas temporales (denominadas *tablas de transición*) conteniendo todas las filas afectadas. Las tablas de transición no se pueden usar con los disparadores **before**, pero sí con los **after**, independientemente de si son disparadores de instrucciones (**statement**) o de filas (**row**).

Una instrucción SQL única se puede usar entonces para realizar varias acciones en términos de las tablas de transición.

Volviendo al ejemplo de inventario del almacén, supóngase que se tienen las siguientes relaciones:

- *inventario*(*producto*, *nivel*), que denota la cantidad actual (número/peso/volumen) del producto en el almacén.
- *nivelmínimo*(*producto*, *nivel*), que denota la cantidad mínima a mantener de cada producto.
- *nuevopedido*(*producto*, *cantidad*), que denota la cantidad de producto a pedir cuando su nivel cae por debajo del mínimo.
- *pedidos*(*producto*, *cantidad*), que denota la cantidad de producto a pedir.

Se puede usar entonces el disparador mostrado en la Figura 6.4 para solicitar un nuevo pedido del producto.

Obsérvese que se ha sido cuidadoso a la hora de realizar un pedido sólo cuando su cantidad caiga desde un nivel superior al mínimo a otro inferior. Si sólo se comprobase si el nuevo valor después de la actualización está por debajo del mínimo, se podría realizar erróneamente un pedido cuando el producto ya se ha pedido.

Muchos sistemas de bases de datos proporcionan implementaciones no estándar de disparadores, o implementan sólo algunas de las características de los disparadores. Por ejemplo, muchos de los sistemas de bases de datos no implementan la cláusula **before**, y usan la palabra clave **on** en lugar de **after**. Puede que no implementen la cláusula **referencing**. En su lugar, pueden especificar tablas de transición usando las palabras clave **inserted** o **deleted**. La Figura 6.5 ilustra cómo se escribiría el disparador de los descubiertos de las cuentas en SQLServer de Microsoft. Consúltese el manual del sistema de bases de datos que se esté usando para obtener más información sobre las características de los disparadores que soporta.

6.4.3. Cuándo no deben usarse los disparadores

Hay muchos buenos usos de los disparadores, como los que se acaban de ver en el Apartado 6.4.2, pero algunos usos se manejan mejor con otras técnicas. Por ejemplo, anteriormente, los diseñadores de sistemas usaban los disparadores para realizar resúmenes de datos. Por ejemplo, usaban disparadores bajo la inserción, borrado o actualización de una relación *empleado* que contiene los atributos *sueldo* y *dept* para mantener el sueldo total de cada departamento. Sin embargo, muchos sistemas de bases de datos actuales soportan las vistas materializadas (véase el Apartado 3.5.1), que proporcionan una forma mucho más sencilla de mantener los datos de resumen. Los diseñadores también usaban ampliamente los disparadores para las réplicas de las bases de datos; usaban disparadores bajo la inserción, borrado o actualización de las relaciones para registrar los cambios en relaciones **cambio** o **delta**. Un proceso separado copiaba los cambios a la réplica (copia) de la base de datos, y el sistema ejecutaba los cambios sobre la réplica. Sin embargo, los sistemas de bases de datos modernos proporcionan características incorporadas para la réplica de bases de datos, haciendo innecesarios a los disparadores para la réplica en la mayoría de los casos.

De hecho, muchas aplicaciones de disparadores, incluyendo el ejemplo de descubiertos, se pueden sustituir por las características de «encapsulación» introducidas en SQL:1999. La encapsulación se puede usar para asegurar que las actualizaciones del atributo *saldo* de *cuenta* se hagan sólo mediante un procedimiento especial. Este procedimiento comprobaría un saldo negativo y realizaría las acciones de disparador de descubiertos. Las encapsulaciones pueden reemplazar el disparador de nuevos pedidos de forma similar.

```
create trigger nuevopedido after update of cantidad on inventario
referencing old row as ofila, new row as nfila
for each row
when nfila.nivel <= (select nivel
                     from nivelmínimo
                     where nivelmínimo.producto = ofila.producto)
and ofila.nivel <= (select nivel
                     from nivelmínimo
                     where nivelmínimo.producto = ofila.producto)
begin
    insert into pedidos
        (select producto, cantidad
         from nuevopedido
         where nuevopedido.producto= ofila.producto);
end
```

FIGURA 6.4. Ejemplo de un disparador para hacer un nuevo pedido de un producto.

```
define trigger descubierto on cuenta
for update
as
if nfila.saldo < 0
begin
      insert into prestatario
            (select nombre-cliente, número-cuenta
             from impositor, inserted
             where inserted.número-cuenta = impositor.número-cuenta)
      insert into préstamo values
            (inserted.número-cuenta, inserted.nombre-sucursal, – inserted.saldo)
      update cuenta set saldo = 0
            from cuenta, inserted
            where cuenta.número-cuenta = inserted.número-cuenta))
end
```

FIGURA 6.5. Ejemplo de disparador en la sintaxis de SQLServer de Microsoft.

Los disparadores se deberían escribir con sumo cuidado, dado que un error de un disparador detectado en tiempo de ejecución causa el fallo de la instrucción de inserción, borrado o actualización que inició el disparador. En el peor de los casos esto podría dar lugar a una cadena infinita de disparos. Por ejemplo, supóngase que un disparador de inserción sobre una relación realice otra (nueva) inserción sobre la misma relación. La acción de inserción dispara otra acción de inserción, y así hasta el infinito. Generalmente, los sistemas de bases de datos limitan la longitud de las cadenas de disparadores (por ejemplo, hasta 16 o 32), y consideran que las cadenas mayores son erróneas.

Los disparadores (desencadenadores en terminología de Microsoft) se denominan a veces *reglas* o *reglas activas*, pero no se deben confundir con las reglas Datalog (véase el Apartado 5.2), que son realmente definiciones de vistas.

La palabra clave **new** utilizada delante de *T.saldo* indica que debe utilizarse el valor de *T.saldo* posterior a la actualización; si se omite, se utilizará el valor previo a la actualización.

6.5. SEGURIDAD Y AUTORIZACIÓN

Los datos guardados en la base de datos deben estar protegidos contra los accesos no autorizados, de la destrucción o alteración malintencionadas además de la introducción accidental de inconsistencias que evitan las restricciones de integridad. En este apartado se examina el modo en que se pueden utilizar mal los datos o hacerlos inconsistentes de manera intencionada. Se presentan luego mecanismos para protegerse de dichas incidencias.

6.5.1. Violaciones de la seguridad

Entre las formas de acceso malintencionado se encuentran:

- La lectura no autorizada de los datos (robo de información)
- La modificación no autorizada de los datos
- La destrucción no autorizada de los datos

La **seguridad de las bases de datos** se refiere a la protección frente a accesos malintencionados. No es posible la protección absoluta de la base de datos contra el uso malintencionado, pero se puede elevar lo suficiente el coste para quien lo comete como para disuadir la mayor parte, si no la totalidad, de los intentos de tener acceso a la base de datos sin la autorización adecuada.

Para proteger la base de datos hay que adoptar medidas de seguridad en varios niveles:

- **Sistema de bases de datos.** Puede que algunos usuarios del sistema de bases de datos sólo estén autorizados a tener acceso a una parte limitada de la base de datos. Puede que otros usuarios estén autorizados a formular consultas pero tengan prohibido modificar los datos. Es responsabilidad del sistema de bases de datos asegurarse de que no se violen estas restricciones de autorización.
- **Sistema operativo.** Independientemente de lo seguro que pueda ser el sistema de bases de datos, la debilidad de la seguridad del sistema operativo puede servir como medio para el acceso no autorizado a la base de datos.
- **Red.** Dado que casi todos los sistemas de bases de datos permiten el acceso remoto mediante terminales o redes, la seguridad en el nivel del software de la red es tan importante como la seguridad física, tanto en Internet como en las redes privadas de las empresas.

- **Físico.** Los sitios que contienen los sistemas informáticos deben estar protegidos físicamente contra la entrada de intrusos.
- **Humano.** Los usuarios deben ser autorizados cuidadosamente para reducir la posibilidad de que alguno de ellos dé acceso a intrusos a cambio de sobornos u otros favores.

Debe conservarse la seguridad en todos estos niveles si hay que asegurar la seguridad de la base de datos. La debilidad de los niveles bajos de seguridad (físico o humano) permite burlar las medidas de seguridad estrictas de niveles superiores (base de datos).

En el resto de este apartado se aborda la seguridad en el nivel del sistema de bases de datos. La seguridad en los niveles físico y humano, aunque importante, cae fuera del alcance de este texto.

La seguridad dentro del sistema operativo se aplica en varios niveles, que van desde las contraseñas para el acceso al sistema hasta el aislamiento de los procesos concurrentes que se ejecutan en el sistema. El sistema de archivos también proporciona algún nivel de protección. Las notas bibliográficas hacen referencia al tratamiento de estos temas en los textos sobre sistemas operativos. Finalmente, la seguridad en el nivel de la red ha logrado un amplio reconocimiento a medida que Internet ha pasado de ser una plataforma para la investigación académica a convertirse en la base del comercio electrónico internacional. Las notas bibliográficas muestran el tratamiento dado por los libros de texto a los principios básicos de la seguridad de las redes. Se presenta la discusión sobre la seguridad en términos del modelo relacional de datos, aunque los conceptos de este capítulo son igualmente aplicables a todos los modelos de datos.

6.5.2. Autorizaciones

Los usuarios pueden tener varios tipos de autorización para diferentes partes de la base de datos. Entre ellas están las siguientes:

- **La autorización de lectura** permite la lectura de los datos, pero no su modificación.
- **La autorización de inserción** permite la inserción de datos nuevos, pero no la modificación de los existentes.
- **La autorización de actualización** permite la modificación de los datos, pero no su borrado.
- **La autorización de borrado** permite el borrado de los datos.

Los usuarios pueden recibir todos los tipos de autorización, ninguno de ellos o una combinación determinada de los mismos.

Además de estas formas de autorización para el acceso a los datos, los usuarios pueden recibir autorización para modificar el esquema de la base de datos:

- **La autorización de índices** permite la creación y borrado de índices.
- **La autorización de recursos** permite la creación de relaciones nuevas.
- **La autorización de alteración** permite el añadido o el borrado de atributos de las relaciones.
- **La autorización de eliminación** permite el borrado de relaciones.

Las autorizaciones de **eliminación** y de **borrado** se diferencian en que la autorización de borrado sólo permite el borrado de tuplas. Si un usuario borra todas las tuplas de una relación, la relación sigue existiendo, pero está vacía. Si se elimina una relación, deja de existir.

La capacidad de crear nuevas relaciones queda regulada mediante la autorización de **recursos**. El usuario con la autorización de **recursos** que crea una relación nueva recibe automáticamente todos los privilegios sobre la misma.

La autorización de **índices** puede parecer innecesaria, dado que la creación o borrado de un índice no afecta a los datos de las relaciones. Más bien, los índices son una estructura para las mejoras de rendimiento. Sin embargo, los índices también ocupan espacio y se exige que todas las modificaciones de las bases de datos actualicen los índices. Si se concediera a todos los usuarios la autorización de **índices**, los que llevaran a cabo actualizaciones estarían tentados de borrar los índices, mientras que los que formularan consultas estarían tentados de crear numerosos índices. Para permitir al *administrador de la base de datos* que regule el uso de los recursos del sistema es necesario tratar la creación de índices como un privilegio.

La forma superior de autoridad es la concedida al administrador de la base de datos. El administrador de la base de datos puede autorizar usuarios nuevos, reestructurar la base de datos, etcétera. Esta forma de autorización es análoga a la proporcionada al **superusuario** u operador del sistema operativo.

6.5.3. Autorizaciones y vistas

En el Capítulo 3 se introdujo el concepto de las *vistas* como medio de proporcionar a un usuario un modelo personalizado de la base de datos. Una vista puede ocultar los datos que un usuario no necesita ver. La capacidad de las vistas para ocultar datos sirve para simplificar el uso del sistema y para mejorar la seguridad. El uso del sistema se simplifica porque se permite al usuario restringir su atención a los datos de interés. Aunque puede que se niegue el acceso directo a una relación, puede que se le permita el acceso a parte de esa relación mediante una vista. Por tanto, se puede utilizar una combinación de seguridad en el nivel relacional y en el nivel de las vistas para limitar el acceso de un usuario precisamente a los datos que necesita.

En el ejemplo bancario considérese un empleado que necesita saber los nombres de todos los clientes que tienen un préstamo en cada sucursal. Este empleado no está autorizado a ver la información concerniente a los préstamos concretos que pueda tener cada cliente. Por tanto, se le debe negar el acceso directo a la relación *préstamo*. Pero si va a tener acceso a la información necesaria se le debe conceder acceso a la vista *cliente-préstamo*, que consiste sólo en los nombres de los clientes y las sucursales en los que tienen un préstamo. Esta vista se puede definir en SQL de la manera siguiente:

```
create view cliente-préstamo as
    (select nombre-sucursal, nombre-cliente
    from prestatario, préstamo
    where prestatario.número-préstamo
            = préstamo.número-préstamo)
```

Supóngase que el empleado formula la siguiente consulta SQL:

```
select *
from préstamo-cliente
```

Evidentemente, el empleado está autorizado a ver el resultado de esta consulta. Sin embargo, cuando el procesador de consultas traduce la consulta en una consulta sobre las relaciones reales de la base de datos, se obtiene una consulta sobre *prestatario* y *préstamo*. Por tanto, se debe comprobar la autorización de la consulta del empleado antes de que comience el procesamiento de la misma.

La creación de vistas no necesita la autorización de **recursos**. El usuario que crea una vista no recibe necesariamente todos los privilegios sobre la misma. Ese usuario sólo recibe los privilegios que no proporcionan autorizaciones adicionales respecto de las que ya posee. Por ejemplo, un usuario no puede recibir la autorización de **actualización** sobre una vista sin tener la autorización de **actualización** sobre las relaciones utilizadas para definir la vista. Si un usuario crea una vista sobre la que no se puede conceder ninguna autorización, se deniega la solicitud de creación de la vista. En el ejemplo de la vista *cliente-préstamo*, el creador de la vista debe tener autorización de **lectura** sobre las relaciones *prestatario* y *préstamo*.

6.5.4. Concesión de privilegios

El usuario al que se le ha concedido alguna forma de autorización puede ser autorizado a transmitir esa autorización a otros usuarios. Sin embargo, hay que tener cuidado con el modo en que se puede transmitir la autorización entre los usuarios para asegurar que la misma pueda retirarse en el futuro.

Considérese, a modo de ejemplo, la concesión de la autorización de actualización sobre la relación *préstamo* de la base de datos bancaria. Supóngase que, inicialmente, el administrador de la base de datos concede autorización de actualización sobre *préstamo* a los usuarios U_1, U_2 y U_3, que, a su vez, pueden transmitirla a otros usuarios. La transmisión de la autorización de un usuario a otro puede representarse mediante un **grafo de autorización**. Los nodos de este grafo son los usuarios. Se incluye un arco $U_i \rightarrow U_j$ en el grafo si el usuario U_i concede la autorización de actualización sobre *préstamo* a U_j. La raíz del grafo es el administrador de la base de datos. En la Figura 6.6 aparece un grafo sencillo. Obsérvese que tanto U_1 como U_2 conceden autorización al usuario U_5; sólo U_1 concede autorización a U_4.

Un usuario tiene autorización *si y sólo si* hay un camino desde la raíz del grafo de autorización (es decir, el nodo que representa al administrador de la base de datos) hasta el nodo que representa a ese usuario.

Un par de usuarios taimados podría intentar eludir las reglas de retirada de autorizaciones concediéndose autorización mutuamente, como se muestra en la Figura 6.7a. Si el administrador de la base de datos retira la autorización a U_2, U_2 la conserva mediante U_3, como se muestra en la Figura 6.7b. Si a continuación se retira la autorización a U_3, U_3 parece conservarla mediante U_2, como se muestra en la Figura 6.7.c. Sin embargo, cuando el administrador retira la autorización a U_3, los arcos de U_3 a U_2 y de U_2 a U_3 ya no forman parte de ningún camino que comience en el administrador de la base de datos. Hace falta que todos los arcos de un grafo de autorización sean parte de algún camino que comience en el administrador de la base de datos. Estos arcos se borran; el grafo de autorización resultante queda como se muestra en la Figura 6.8.

Se requiere que todos los arcos de un grafo de autorización sean parte de algún camino que se origine en el administrador de la base de datos. Los arcos entre U_2 y U_3 se borran, y el grafo de autorización es como el mostrado en la Figura 6.8.

6.5.5. El concepto de papel

Considérese un banco que tiene muchos cajeros. Cada cajero debe tener los mismos tipos de autorizaciones

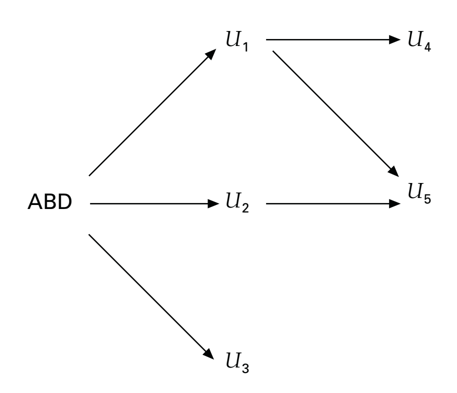


FIGURA 6.6. Grafo de concesión de autorizaciones.

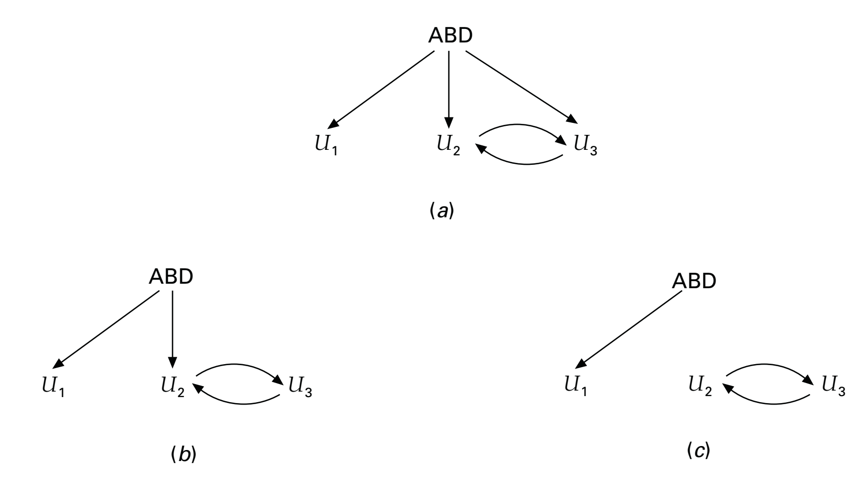


FIGURA 6.7. Intento de eludir la retirada de autorizaciones.

para el mismo conjunto de relaciones. Cada vez que se contrata un nuevo cajero hay que darle todas esas autorizaciones individualmente.

Un esquema mejor sería especificar las autorizaciones que deben tener los cajeros e identificar separadamente cuáles de los usuarios de la base de datos son cajeros. El sistema puede usar estas dos piezas de información para determinar las autorizaciones de cada persona que sea un cajero. Cuando se contrate a una nueva persona como cajero, se le debe asignar un identificador de usuario e identificarle como cajero. Ya no es necesario especificar permisos individuales para los cajeros.

Los **papeles** capturan este esquema. En la base de datos se crea un conjunto de papeles. Las autorizaciones se conceden a los papeles, de igual modo que se conceden a usuarios individuales. Se concede un conjunto de papeles a cada usuario de la base de datos (que puede ser vacío) para los que está autorizado.

En el ejemplo bancario, algunos ejemplos de papeles serían *cajero, gestor-sucursal, auditor* y *administrador-del-sistema*.

Una alternativa menos preferible sería crear un identificador de usuario *cajero* y permitir que cada cajero se conectase a la base de datos usando este identificador. El problema con este esquema es que no sería posible identificar exactamente al cajero que ha realizado una determinada transacción, conduciendo a problemas de seguridad. El uso de los papeles tiene la ventaja de requerir a los usuarios que se conecten con su propio identificador de usuario.

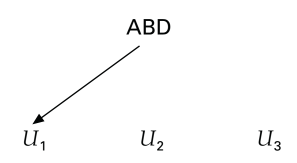


FIGURA 6.8. Grafo de autorización.

Cualquier autorización que se pueda conceder a un usuario se puede conceder a un papel. Los papeles se conceden a los usuarios al igual que las autorizaciones. Y, como otras autorizaciones, un usuario también puede conceder autorización para conceder un papel particular a otros. Así, los gestores de las sucursales pueden tener autorización de concesión para conceder el papel *cajero*.

6.5.6. Trazas de auditoría

Muchas aplicaciones de bases de datos seguras requieren que se mantenga una **traza de auditoría**. Una traza de auditoría es un registro histórico de todos los cambios (inserciones, borrados o actualizaciones) de la base de datos, junto con información sobre el usuario que realizó el cambio y en qué momento.

La traza de auditoría ayuda a la seguridad de formas diferentes. Por ejemplo, si el saldo de una cuenta es incorrecto, el banco desearía revisar todas las actualizaciones realizadas sobre la cuenta para encontrar las actualizaciones incorrectas (o fraudulentas), así como las personas que realizaron los cambios. El banco podría entonces usar la traza de auditoría para revisar todas las actualizaciones realizadas por estas personas para encontrar las actualizaciones incorrectas o fraudulentas.

Es posible crear una traza de auditoría definiendo disparadores adecuados sobre las actualizaciones de las relaciones (usando variables definidas del sistema que identifican el nombre de usuario y la hora). Sin embargo, muchos sistemas de bases de datos proporcionan mecanismos incorporados para crear trazas de auditoría que son más convenientes de usar. Los detalles de la creación de las trazas de auditoría varían entre los sistemas de bases de datos y se deben consultar los manuales para los detalles.

6.6. AUTORIZACIÓN EN SQL

El lenguaje SQL ofrece un mecanismo bastante potente para la definición de autorizaciones. En este apartado se describen estos mecanismos y sus limitaciones.

6.6.1. Privilegios en SQL

La norma SQL incluye los privilegios **delete**, **insert**, **select** y **update** El privilegio **select** se corresponde con el privilegio de **lectura**. SQL también incluye un privilegio **references** que permite a un usuario o papel declarar claves externas al crear relaciones. Si la relación que se va a crear incluye una clave externa que hace referencia a atributos de otra relación, el usuario o papel debe haber recibido el privilegio **references** sobre esos atributos. El motivo de que el privilegio **references** sea una característica útil es algo sutil y se explica más adelante en este mismo apartado.

El lenguaje de definición de datos de SQL incluye órdenes para conceder y retirar privilegios. La instrucción **grant** se utiliza para conferir autorizaciones. La forma básica de esta instrucción es la siguiente:

grant <lista de privilegios> **on** <nombre de relación o de lista> **to** <lista de usuarios/papeles>

La *lista de privilegios* permite la concesión de varios privilegios con una sola orden.

La siguiente instrucción **grant** concede a los usuarios U_1, U_2 y U_3 la autorización **select** sobre la relación *sucursal*:

grant select on *sucursal* **to** U_1, U_2, U_3

La autorización **update** puede concederse sobre todos los atributos de la relación o sólo sobre algunos. Si se incluye la autorización **update** en una instrucción **grant**, la lista de atributos sobre los que se va a conceder la autorización **update** opcionalmente aparece entre paréntesis justo después de la palabra clave **update**. Si se omite la lista de atributos se concede el privilegio **update** sobre todos los atributos de la relación.

La siguiente instrucción **grant** concede a los usuarios U_1, U_2 y U_3 la autorización **update** sobre el atributo *importe* de la relación *préstamo*:

grant update (*importe*) **on** *préstamo* **to** U_1, U_2, U_3

En SQL el privilegio **insert** también puede especificar una lista de atributos; cualquier inserción en la relación debe especificar sólo esos atributos y al resto de los atributos se les conceden valores predeterminados (si hay alguno definido para ellos) o se les da el valor nulo.

El privilegio SQL **references** se concede sobre atributos concretos de manera parecida a la mostrada para el privilegio **update**. La siguiente instrucción **grant** permite al usuario U_1 crear relaciones que hagan referencia a la clave *nombre-sucursal* de la relación *sucursal* como si fuera una clave externa:

grant references (*nombre-sucursal*) **on** *sucursal* **to** U_1

En un principio puede parecer que no hay ningún motivo para evitar que los usuarios creen claves externas que hagan referencia a otra relación. Sin embargo, hay que recordar del Apartado 6.2 que las restricciones de las claves externas limitan las operaciones de borrado y de actualización sobre la operación a la que hacen referencia. En el ejemplo anterior, si U_1 crea una clave externa en una relación *r* que haga referencia al atributo *nombre-sucursal* de la relación *sucursal* y luego inserta en *r* una tupla correspondiente a la sucursal de Navacerrada, ya no es posible borrar la sucursal de Navacerrada de la relación *sucursal* sin modificar también la relación *r*. Por tanto, la definición de una clave externa por U_1 limita la actividad futura de los demás usuarios; por tanto, el privilegio **references** es necesario.

El privilegio **all privileges** puede utilizarse como una forma abreviada de todos los privilegios que se pueden conceder. De manera parecida, el nombre de usuario **public** hace referencia a todos los usuarios presentes y futuros del sistema. SQL incluye también un privilegio **usage** que autoriza a un usuario a utilizar un dominio especificado (recuérdese que el dominio se corresponde con el concepto de tipo de un lenguaje de programación y puede ser definido por el usuario).

6.6.2. Papeles

Los papeles se pueden crear en SQL:1999 como sigue

create role *cajero*

Se pueden conceder privilegios a los papeles al igual que a los usuarios, como se ilustra en la siguiente instrucción

grant select on *cuenta*
to *cajero*

Los papeles se pueden asignar a los usuarios, así como a otros papeles, como muestran estas instrucciones.

grant *cajero* **to** juan
create role *gestor*
grant *cajero* **to** *gestor*
grant *gestor* **to** maría

Los privilegios de un usuario o papel consisten en:

- Todos los privilegios concedidos directamente al usuario o papel.
- Todos los privilegios concedidos a papeles que se hayan concedido al usuario o papel.

Nótese que puede haber una cadena de papeles; por ejemplo, el papel *empleado* se puede conceder a todos los *cajeros*. A su vez, el papel *cajero* se concede a todos los *gestores*. Así, el papel *gestor* hereda todos los privilegios concedidos a los papeles *empleados* y *cajeros* además de los privilegios concedidos directamente a *gestor*.

6.6.3. El privilegio de conceder privilegios

Un usuario o papel al que se le concede un privilegio no está autorizado de manera predeterminada a concedérselo a otros usuarios o papeles. Si se desea conceder un privilegio a un usuario y permitirle que lo transmita a otros usuarios hay que añadir la cláusula **with grant option** a la orden **grant** correspondiente. Por ejemplo, si se desea conceder a U_1 el privilegio **select** sobre *sucursal* y que pueda transmitirlo a otros, hay que escribir

grant select on *sucursal* **to** U_1 **with grant option**

Para retirar una autorización se utiliza la instrucción **revoke**. Adopta una forma casi idéntica a la de **grant**:

revoke <lista de privilegios> **on** <nombre de relación o de vista>
from <lista de usuarios o papeles> [**restrict** | **cascade**]

Por tanto, para retirar los privilegios que se han concedido con anterioridad hay que escribir

revoke select on *sucursal* **from** U_1, U_2, U_3
revoke update (*importe*) **on** *préstamo* **from** U_1, U_2, U_3
revoke references (*nombre-sucursal*) **on** *sucursal* **from** U_1

Como se vio en el Apartado 6.5.4, la retirada de un privilegio a un usuario o papel puede hacer que otros usuarios o papeles también lo pierdan. Este comportamiento se denomina *retirada en cadena*. En la mayoría de los sistemas de bases de datos, la retirada en cascada es el comportamiento predeterminado; por tanto, la palabra clave **cascade** se puede omitir como se ha hecho en los ejemplos anteriores. La instrucción **revoke** también puede especificar **restrict**:

revoke select on *sucursal* **from** U_1, U_2, U_3 **restrict**

En este caso se devuelve un error si hay alguna retirada de privilegios en cadena y en este caso la acción de retirada de privilegios no se lleva a cabo. La siguiente instrucción **revoke** sólo anula la opción de concesión **grant** y no el propio privilegio **select**.

revoke grant option for select on *sucursal* **from** U_1

6.6.4. Otras características

El creador de un objeto (relación, vista o papel) obtiene todos los privilegios sobre el objeto, incluyendo el privilegio de conceder privilegios a otros.

La norma SQL especifica un mecanismo primitivo de autorización para el esquema de la base de datos: sólo el propietario del esquema puede ejecutar cualquier modificación del esquema. Por tanto, las modificaciones del esquema —como la creación o borrado de las relaciones, la adición o eliminación de atributos de las relaciones y la adición o eliminación de índices— sólo pueden ser ejecutadas por el propietario del mismo. Varias implementaciones de las bases de datos tienen mecanismos de autorización más potentes para los esquemas de las bases de datos, parecidos a los discutidos anteriormente, pero esos mecanismos no son estándar.

6.6.5. Limitaciones de la autorización SQL

Las normas SQL actuales de autorización tienen algunas deficiencias. Por ejemplo, supóngase que se desea que todos los estudiantes sean capaces de ver sus propias notas, pero no las del resto. La autorización debe estar en el nivel de las tuplas, lo cual no es posible en los estándares de autorización de SQL.

Más aún, con el crecimiento de Web, los accesos a bases de datos vienen principalmente de los servidores de aplicaciones Web. Los usuarios finales puede que no tengan identificadores de usuario individuales en la base de datos y realmente puede que haya sólo un único identificador de usuario en la base de datos que corresponda a todos los usuarios del servidor de aplicaciones.

La tarea de la autorización recae entonces sobre el servidor de aplicaciones; el esquema completo de autorización de SQL se omite. El beneficio es que la aplicación puede implementar las autorizaciones de grano fino, como las de las tuplas individuales. Estos son los problemas:

- El código para comprobar la autorización se entremezcla con el resto del código de la aplicación.
- La implementación de la autorización mediante código de aplicación, en lugar de realizarlo declarativamente con SQL, hace que sea difícil asegurar la ausencia de agujeros de seguridad. Debido a un descuido, uno de los programas de aplicación podría no comprobar la autorización, permitiendo el acceso de usuarios no autorizados a datos confidenciales. La verificación de que todos los programas de aplicación realizan todas las comprobaciones de autorización implica la lectura de todo el código del servidor de la aplicación, una tarea formidable en un gran sistema.

6.7. CIFRADO Y AUTENTICACIÓN

Puede que las diferentes provisiones para la autorización que haga un sistema de bases de datos no proporcionen suficiente protección para los datos extremadamente delicados. En tales casos se pueden **cifrar** los datos. Los datos cifrados no se pueden leer a menos que el lector sepa la manera de **descifrarlos** (*descodificarlos*). El cifrado también forma la base de los buenos esquemas para la autenticación de usuarios en una base de datos.

6.7.1. Técnicas de cifrado

Hay un enorme número de técnicas para el cifrado de los datos. Puede que las técnicas de cifrado sencillas no proporcionen la seguridad adecuada, dado que puede ser sencillo para un usuario no autorizado romper el código. Como ejemplo de una técnica de cifrado débil considérese la sustitución de cada carácter por el siguiente en el alfabeto. Así,

Navacerrada

se transforma en

Ñbwbdfssbeb

Si un usuario no autorizado sólo lee «Ñbwbdfssbeb» probablemente no tenga la información suficiente para romper el código. Sin embargo, si el intruso ve un gran número de nombres de sucursales codificados puede utilizar los datos estadísticos referentes a la frecuencia relativa de los caracteres para averiguar el tipo de sustitución realizado (por ejemplo, en inglés la *E* es la más frecuente, seguida por *T, A, O, N, I, …*).

Una buena técnica de cifrado tiene las propiedades siguientes:

- Es relativamente sencillo para los usuarios autorizados cifrar y descifrar los datos.
- El esquema de cifrado no depende de lo poco conocido que sea el algoritmo, sino más bien de un parámetro del algoritmo denominado *clave de cifrado*.
- Es extremadamente difícil para un intruso determinar la clave de cifrado.

Un enfoque, la *norma de cifrado de datos* (*Data Encription Standard*, DES) realiza una sustitución de caracteres y una reordenación de los mismos en función de una clave de cifrado. Para que este esquema funcione los usuarios autorizados deben proveerse de la clave de cifrado mediante un mecanismo seguro. Este requisito es una debilidad importante, dado que el esquema no es más seguro que el mecanismo por el que se transmite la clave de cifrado. La norma DES se reafirmó en 1983, 1987 y de nuevo en 1993. Sin embargo, la debilidad de DES se reconoció en 1993 cuando se necesitó una nueva norma denominada *norma de cifrado avanzado* (*Advanced Encription Standard*, AES). En el año 2000, el **algoritmo Rijndael** (denominado así por sus inventores, V. Rijmen y J. Daemen) se seleccionó para ser la norma AES. Este algoritmo se eligió por su nivel significativamente más fuerte de seguridad y su facilidad relativa de implementación en los sistemas informáticos actuales, así como en dispositivos como tarjetas inteligentes. Al igual que la norma DES, el algoritmo Rijndael es un algoritmo de clave compartida (o clave simétrica) en el que los usuarios autorizados comparten una clave.

El **cifrado de clave pública** es un esquema alternativo que evita parte de los problemas que se afrontan con DES. Se basa en dos claves; una *clave pública* y una *clave privada*. Cada usuario U_i tiene una clave pública C_i y una clave privada D_i. Todas las claves públicas están publicadas: cualquiera puede verlas. Cada clave privada sólo la conoce el usuario al que pertenece. Si el usuario U_1 desea guardar datos cifrados, los cifra utilizando la clave pública C_1. Descifrarlos requiere la clave privada D_1.

Como la clave de cifrado de cada usuario es pública es posible intercambiar información de manera segura utilizando este esquema. Si el usuario U_1 desea compartir los datos con U_2 los codifica utilizando E_2, la clave pública de U_2. Dado que sólo el usuario U_2 conoce la manera de descifrar los datos, la información se transmite de manera segura.

Para que el cifrado de clave pública funcione debe haber un esquema de cifrado que pueda hacerse público sin permitir a la gente descubrir el esquema de descifrado. En otros términos, debe ser difícil deducir la clave privada dada la clave pública. Existe un esquema de ese tipo y se basa en lo siguiente:

- Hay un algoritmo eficiente para comprobar si un número es primo.
- No se conoce ningún algoritmo eficiente para encontrar los factores primos de un número.

Para los fines de este esquema los datos se tratan como una colección de enteros. Se crea una clave pública calculando el producto de dos números primos grandes: P_1 y P_2. La clave privada consiste en el par (P_1, P_2) y el algoritmo de descifrado no se puede utilizar con éxito si sólo se conoce el producto P_1P_2. Dado que todo lo que se publica es el producto P_1P_2, un usuario no autorizado debería poder factorizar P_1P_2 para robar datos. Eligiendo P_1 y P_2 suficientemente grandes (por encima de 100 dígitos) se puede hacer el coste de la factorización prohibitivamente elevado (del orden de años

de tiempo de cálculo, incluso en las computadoras más rápidas).

En las notas bibliográficas se hace referencia a los detalles del cifrado de clave pública y la justificación matemática de las propiedades de esta técnica.

Pese a que el cifrado de clave pública que utiliza el esquema descrito es seguro, también es costoso en cuanto a cálculo. Un esquema híbrido utilizado para la comunicación segura es el siguiente: las claves DES se intercambian mediante un esquema de cifrado de clave pública y posteriormente se utiliza el cifrado DES con los datos transmitidos.

6.7.2. Autenticación

La autenticación se refiere a la tarea de verificar la identidad de una persona o software que se conecte a una base de datos. La forma más simple consiste en una contraseña secreta que se debe presentar cuando se abra una conexión a la base de datos.

La autenticación basada en palabras clave se usa ampliamente por los sistemas operativos y bases de datos. Sin embargo, el uso de contraseñas tiene algunos inconvenientes, especialmente en una red. Si un husmeador es capaz de «oler» los datos que circulan por la red, puede ser capaz de encontrar la contraseña que se está enviando por la red. Una vez que el husmeador tiene un usuario y contraseña, se puede conectar a la base de datos pretendiendo que es el usuario legítimo.

Un esquema más seguro es el sistema de **desafío-respuesta**. El sistema de bases de datos envía una cadena de desafío al usuario. El usuario cifra la cadena de desafío usando una contraseña secreta como clave de cifrado y devuelve el resultado. El sistema de bases de datos puede verificar la autenticidad del usuario descifrando la cadena con la mima contraseña secreta, y comparando el resultado con la cadena de desafío original. Este esquema asegura que las contraseñas no circulen por la red.

Los sistemas de clave pública se pueden usar para cifrar en un sistema de desafío-respuesta. El sistema de bases de datos cifra una cadena de desafío usando la clave pública del usuario y lo envía al usuario. Éste descifra la cadena con su clave privada y devuelve el resultado al sistema de bases de datos. El sistema de bases de datos comprueba entonces la respuesta. Este esquema tiene la ventaja añadida de no almacenar la contraseña en la base de datos, donde podría ser vista potencialmente por administradores del sistema.

Otra aplicación interesante de la criptografía está en las **firmas digitales** para verificar la autenticidad de los datos; las firmas digitales desempeñan el papel electrónico de las firmas físicas en los documentos. La clave privada se usa para firmar los datos y los datos firmados se pueden hacer públicos. Cualquiera podría verificarlos con la clave pública, pero nadie podría haber generado los datos codificados sin tener la clave privada. Por tanto, se puede comprobar que los datos fueron creados realmente por la persona que afirma haberlos creado.

Además, las firmas digitales también sirven para asegurar el **rechazo**. Es decir, en el caso de que una persona que creó los datos afirmase más tarde que no lo hizo (el equivalente electrónico de afirmar que no se ha firmado un talón) se puede probar que esa persona ha creado los datos (a menos que haya cedido su clave privada a otros).

6.8. RESUMEN

- Las restricciones de integridad aseguran que las modificaciones realizadas a la base de datos por los usuarios autorizados no den lugar a la pérdida de la consistencia de los datos.
- En capítulos anteriores se tomaron en consideración varios tipos de restricciones, incluyendo la declaración de claves y la declaración de la forma de las relaciones (de varios a varios, de varios a uno, de uno a uno). En este capítulo se han tomado en consideración otros tipos de restricciones y se han discutido mecanismos para asegurar el mantenimiento de estas restricciones.
- Las restricciones de los dominios especifican el conjunto de valores que se pueden asociar con un atributo. Estas restricciones también pueden impedir el uso de valores nulos para atributos concretos.
- Las restricciones de integridad referencial aseguran que un valor que aparece en una relación para un conjunto de atributos dado aparezca también para un conjunto de atributos concreto en otra relación.
- Las restricciones de dominio y las de integridad referencial son relativamente sencillas de verificar. El uso de restricciones más complejas puede provocar una sobrecarga considerable. Se han visto dos maneras de expresar restricciones más generales. Los asertos son expresiones declarativas que manifiestan predicados que deben ser siempre verdaderos.
- Los disparadores definen las acciones que se deben ejecutar automáticamente cuando se producen ciertos eventos. Los disparadores tienen muchos usos, tales como la implementación de reglas de negocio, registro histórico de la auditoría e incluso para realizar acciones fuera del sistema de bases de datos. Aunque los disparadores se añadieron tarde a la norma SQL como parte de SQL:1999, la mayoría de sistemas de bases de datos los han implementado desde hace tiempo.

- Los datos guardados en las bases de datos deben protegerse de los accesos no autorizados, de la destrucción malintencionada o de la alteración y de la introducción accidental de inconsistencias.
- Es más sencilla la protección contra la pérdida accidental de la consistencia de los datos que contra el acceso malintencionado a las bases de datos. La protección total de las bases de datos contra las acciones malintencionadas no es posible, pero puede lograrse que el coste para quienes las cometan sea lo bastante elevado como para disuadir la mayor parte, si no a la totalidad, de los intentos de acceso a la base de datos sin la correspondiente autorización.
- Los usuarios pueden tener varios tipos de autorización para diferentes partes de la base de datos. La autorización es un medio por el que se puede proteger al sistema de bases de datos contra los accesos malintencionados o no autorizados.
- Los usuarios a los que se les haya concedido algún tipo de autorización pueden ser autorizados a transmitir la misma a otros usuarios. Sin embargo, hay que tomar precauciones respecto a la manera en que se pueden transmitir las autorizaciones entre los usuarios si se debe asegurar que las mismas se podrán retirar en el futuro.
- Los papeles facilitan la asignación de un conjunto de privilegios a un usuario de acuerdo al papel que el usuario desempeña en la organización.
- Puede que las diferentes provisiones de autorización de los sistemas de bases de datos no proporcionen la protección suficiente para los datos más delicados. En tales casos se pueden *cifrar* los datos. Sólo quienes conozcan la manera de *descifrar* (descodificar) los datos cifrados podrán leerlos. El cifrado también forma la base de la autenticación segura de los usuarios.

TÉRMINOS DE REPASO

- Aserto
- Autenticación
- Autorización
- Cascada
- Cifrado
- Cifrado de clave pública
- Cifrado de clave secreta
- Cláusula check
- Concesión de privilegios
- Disparador
- Disparadores before y after
- Firma digital
- Grafo de autorización
- Integridad referencial
- Modelo evento-condición-acción
- Niveles de seguridad
- Papeles
- Privilegios
  - — Lectura
  - — Inserción
  - — Actualización
  - — Borrado
  - — Índices
  - — Recursos
  - — Alteración
  - — Eliminación
  - — Concesión
  - — Todos los privilegios
- Rechazo
- Restricción de clave externa
- Restricción de clave primaria
- Restricciones de los dominios
- Restricción de unicidad
- Seguridad de la base de datos
- Sistema de desafío-respuesta
- Variables y tablas de transición

EJERCICIOS

6.1. Complétese la definición del LDD de SQL de la base de datos bancaria de la Figura 6.2 para incluir las relaciones *préstamo* y *prestatario*.

6.2. Considérese la siguiente base de datos relacional:

empleado (<u>*nombre-empleado*</u>, *calle*, *ciudad*)
trabaja (<u>*nombre-empleado*</u>, *nombre-empresa*, *sueldo*)
empresa (<u>*nombre-empresa*</u>, *ciudad*)
jefe (<u>*nombre-empleado*</u>, *nombre-jefe*)

Dese una definición en el LDD de SQL de esta base de datos. Identifíquense las restricciones de integridad

referencial que deban cumplirse e inclúyanse en la definición del LDD.

6.3. Las restricciones de integridad referencial tal y como se han definido en este capítulo implican exactamente a dos relaciones. Considérese una base de datos que incluye las relaciones siguientes:

trabajador-fijo (*nombre*, *despacho*, *teléfono*, *sueldo*)
trabajador-tiempo-parcial (*nombre*, *sueldo-por-hora*)
dirección (*nombre*, *calle*, *ciudad*)

Supóngase que se desea exigir que cada nombre que aparece en *dirección* aparezca en *trabajador-fijo* o en *trabajador-tiempo-parcial*, pero no necesariamente en ambos.

a. Propóngase una sintaxis para expresar esta restricción.
b. Discútanse las acciones que debe realizar el sistema para aplicar una restricción de este tipo.

6.4. SQL permite la dependencia de una clave externa para hacer referencia a la misma relación, como en el ejemplo siguiente:

create table *jefe*
(*nombre-empleado* **char** (20) **not null**
*nombre-jefe***char** (20) **not null**,
primary key *nombre-empleado*,
foreign key (*nombre-jefe*) **references** *jefe*
on delete cascade)

Aquí, *nombre-empleado* es una clave de la tabla *director*, lo que significa que cada empleado tiene como máximo un director. La orden de clave externa exige que cada director sea también empleado. Explíquese exactamente lo que ocurre cuando se borra una tupla de la relación *jefe*.

6.5. Supóngase que hay dos relaciones *r* y *s* tales que la clave externa *B* de *r* hace referencia a la clave primaria *A* de *s*. Descríbase la manera en que puede utilizarse el mecanismo de los disparadores para implementar la opción **on delete cascade** cuando se borra una tupla de *s*.

6.6. Escríbase un aserto para la base de datos bancaria que asegure que el valor del activo de la sucursal de Navacerrada sea igual a la suma de todos los importes prestados por esa sucursal.

6.7. Escríbase un disparador SQL para realizar la siguiente acción: cuando se borre una cuenta, comprobar para cada tenedor de la cuenta si tiene otras cuentas y, si no, borrarlo de la relación impositor.

6.8. Considérese la vista sucursal-cliente definida como sigue:

create view *sucursal-cliente* **as**
select *nombre-sucursal, nombre-cliente*
from *impositor, cuenta*
where *impositor.número-cuenta*
= *cuenta.número-cuenta*

Supóngase que la vista está *materializada*, es decir, la vista se calcula y se almacena. Escríbanse reglas activas para *mantener* la vista, es decir, mantenerla actualizada según las inserciones y borrados en *impositor* o *cuenta*. No hay que preocuparse de las actualizaciones.

6.9. Confecciónese una lista de los problemas de seguridad de un banco. De cada elemento de la lista hay que decir si afecta a la seguridad física, a la personal, a la del sistema operativo o a la de las bases de datos.

6.10. Utilizando las relaciones de la base de datos bancaria de ejemplo escríbase una expresión SQL para definir las vistas siguientes:

a. Una vista que contenga los números de cuenta y los nombres de los clientes (pero no los saldos) de todas las cuentas de la sucursal de El Escorial.
b. Una vista que contenga el nombre y la dirección de todos los clientes que tengan cuenta en el banco pero no tengan ningún préstamo.
c. Una vista que contenga el nombre y el saldo medio de la cuenta de cada cliente de la sucursal de Collado Villalba.

6.11. Para cada una de las vistas definidas en el Ejercicio 6.10 explíquese la manera en que deben realizarse las actualizaciones (si es que se deben permitir). *Sugerencia:* véase la discusión de las vistas en el Capítulo 3.

6.12. En el Capítulo 3 se describió el uso de las vistas para facilitar el acceso a la base de datos de los usuarios que sólo necesiten ver una parte de la misma. En este capítulo se ha descrito el uso de las vistas como mecanismo de seguridad. ¿Pueden entrar en conflicto estas dos finalidades de las vistas? Explíquese la respuesta.

6.13. ¿Cuál es la finalidad de tener categorías diferentes para la autorización de índices y para la de recursos?

6.14. Los sistemas de bases de datos que guardan cada relación en un archivo diferente del sistema operativo pueden utilizar los esquemas de seguridad y de autorización del sistema operativo en lugar de definir un esquema especial propio. Discútanse las ventajas e inconvenientes de ese enfoque.

6.15. ¿Cuáles son las dos ventajas de cifrar los datos guardados en una base de datos?

6.16. Quizás los elementos de datos más importantes de cualquier sistema de bases de datos sean las contraseñas que controlan el acceso a la base de datos. Propóngase un esquema para guardar de manera segura las contraseñas. Asegúrese de que el esquema permita al sistema comprobar las contraseñas proporcionadas por los usuarios que intenten iniciar una sesión en el mismo.

NOTAS BIBLIOGRÁFICAS

En Hammer y McLeod [1975], Stonebraker [1975], Eswaran y Chamberlin [1975], Schmid y Swenson [1975] y en Codd [1979] se ofrecen discusiones sobre las restricciones de integridad en el modelo relacional. Las propuestas originales de SQL para los asertos y los disparadores se discuten en Astrahan et al. [1976], Chamberlin et al. [1976] y en Chamberlin et al. [1981]. Véanse las notas bibliográficas del Capítulo 4 para obtener referencias de las normas SQL.

Las discusiones referentes al mantenimiento eficiente y a la comprobación de los asertos de integridad semántica se ofrecen en Hammer y Sarin [1978], Badal y Popek [1979], Bernstein et al. [1980a], Hsu e Imielinski [1985], McCune y Henschen [1989] y Chomicki [1992a]. Una alternativa al uso de comprobaciones de integridad en tiempo de ejecución es certificar la corrección de los programas que tienen acceso a la base de datos. Este enfoque se discute en Sheard y Stemple [1989].

Las **bases de datos activas** son bases de datos que soportan disparadores y otros mecanismos que permiten a la base de datos realizar acciones cuando sucedan eventos. En McCarthy y Dayal [1989] se discute la arquitectura de un sistema de bases de datos activas basado en el formalismo evento-condición-acción. Widom y Finkelstein [1990] describen la arquitectura de un sistema de reglas basado en reglas orientadas a conjuntos; la implementación de un sistema de reglas en el sistema de bases de datos extensibles Starbust se presenta en Widom et al. [1991]. Considérese un mecanismo de ejecución que permita la elección no determinista de la regla a ejecutar a continuación. Se dice que un sistema de reglas es **confluente** si, independientemente de la regla elegida, el estado final es el mismo. Los aspectos de terminación, no determinismo y confluencia de los sistemas de reglas se discuten en Aiken et al. [1995].

Los aspectos de la seguridad de los sistemas informáticos en general se discuten en Bell y LaPadula [1976] y en el Departamento de Defensa de EE.UU. [U.S. Dept. of Defense 1985]. Los aspectos de la seguridad de SQL se pueden encontrar en las normas de SQL y en los libros de texto sobre SQL referenciados en las notas bibliográficas del Capítulo 4.

Stonebraker y Wong [1974] estudian el enfoque de la seguridad de Ingres, que implica la modificación de las consultas de los usuarios para asegurar que los usuarios no tienen acceso a datos para los que no se les ha concedido autorización. Denning y Denning [1979] presentan una visión del estado del arte de la seguridad de las bases de datos.

Los sistemas de bases de datos que pueden producir respuestas erróneas cuando es necesario para el mantenimiento de la seguridad se discuten en Winslett et al. [1994] y en Tendick y Matloff [1994]. El trabajo en la seguridad de las bases de datos relacionales incluye el de Stachour y Thuraisingham [1990], Jajodia y Sandhu [1990], Kogan y Jajodia [1990] y Qian y Lunt [1996]. Los aspectos de la seguridad de los sistemas operativos se discuten en la mayor parte de los textos sobre sistemas operativos, incluyendo a Silberschatz y Galvin [1998].

Stallings [1998] proporciona una descripción en un libro de texto de la criptografía. Daemen y Rijmen [2000] presentan el algoritmo Rjndael. El Departamento de Comercio de EE. UU. [U.S. Dept. of Commerce 1977] presentó la norma para cifrado de datos. El cifrado mediante clave pública se discute en Rivest et al. [1978]. Otras discusiones sobre criptografía incluyen las de Diffie y Hellman [1979], Simmons [1979], Fernández et al. [1981] y Akl [1983].

CAPÍTULO

7 DISEÑO DE BASES DE DATOS RELACIONALES

Este capítulo continúa el estudio de los problemas de diseño de las bases de datos relacionales. En general, el objetivo del diseño de las bases de datos relacionales es la generación de un conjunto de esquemas relacionales que nos permita almacenar la información sin redundancias innecesarias, pero que también nos permita recuperar fácilmente esa información. Un enfoque es el diseño de esquemas que se hallen en una *forma normal* adecuada. Para determinar si el esquema de una relación se halla en una de las formas normales deseables hace falta información adicional sobre la empresa real que ese está modelando con la base de datos. En este capítulo se introduce el concepto de la dependencia funcional. Luego se definirán las formas normales en términos de las dependencias funcionales y otros tipos de dependencias de datos.

7.1. PRIMERA FORMA NORMAL

La primera de las formas normales que se van a estudiar, la **primera forma normal**, impone un requisito muy elemental a las relaciones; a diferencia de las demás formas normales, no exige información adicional como las dependencias funcionales.

Un dominio es **atómico** si se considera que los elementos del dominio son unidades indivisibles. Se dice que el esquema de una relación *R* está en la **primera forma normal** (1FN) si los dominios de todos los atributos de *R* son atómicos.

Un conjunto de nombres es un ejemplo de valor no atómico. Por ejemplo, si el esquema de la relación *empleado* incluyera el atributo *hijos,* los elementos de cuyo dominio son conjuntos de nombres, el esquema no se hallaría en la primera forma normal.

Los atributos compuestos, como el atributo *dirección* con sus atributos componentes *calle* y *ciudad*, tienen también dominios no atómicos.

Se da por supuesto que los enteros son atómicos, por lo que el conjunto de enteros es un dominio atómico; el conjunto de todos los conjuntos de enteros es un dominio no atómico. La diferencia estriba en que normalmente no se considera que los enteros tengan subpartes, pero sí se considera que los tienen los conjuntos de enteros, es decir, los enteros que componen el conjunto. Pero lo importante no es lo que sea el propio dominio, sino el modo en que se utilizan los elementos del dominio en la base de datos.

El dominio de todos los enteros no sería atómico si se considerara que cada entero es una lista ordenada de cifras.

Como ilustración práctica del punto anterior, considérese una organización que asigna a los empleados números de identificación de la manera siguiente: las dos primeras letras especifican el departamento y las cuatro cifras restantes son un número único para el empleado dentro de ese departamento. Ejemplos de estos números pueden ser *IN*0012 y *EE*1127. Estos números de identificación pueden dividirse en unidades menores y, por tanto, no son atómicos. Si el esquema de una relación tuviera un atributo cuyo dominio consistiera en números de identificación codificados como se ha indicado, el esquema no se hallaría en la primera forma normal.

Cuando se utilizan estos números de identificación, se puede averiguar el departamento de cada empleando escribiendo código que analice la estructura de los números de identificación. Ello exige programación adicional y la información queda codificada en el programa de aplicación en vez de en la base de datos. Surgen nuevos problemas si se utilizan estos números de identificación como claves principales: Cada vez que un empleado cambia de departamento hay que cambiar su número de identificación, lo que puede constituir una tarea difícil, o en su defecto el código que interpreta ese número dará un resultado erróneo.

El empleo de atributos con el valor dado por el conjunto puede llevar a diseños con almacenamiento de datos redundantes, lo que, a su vez, puede dar lugar a inconsistencias. Por ejemplo, en lugar de representar la relación entre las cuentas y los clientes como una relación independiente *impositor*, puede que un diseñador de bases de datos esté tentado a almacenar un conjunto de *titulares* con cada cuenta y un conjunto de *cuentas* con cada cliente. Siempre que se cree una cuenta, o se actualice el conjunto de titulares de una cuenta, hay que llevar a cabo la actualización en dos lugares; no llevar a cabo las dos actualizaciones puede dejar la base

de datos en un estado inconsistente. La conservación de sólo uno de estos conjuntos evitaría la información repetida, pero complicaría algunas consultas. También es más complicado tanto escribir consultas con los atributos con el valor dado por el conjunto como razonar sobre ellos.

En este capítulo sólo se toman en consideración dominios atómicos y se da por supuesto que las relaciones están en la primera forma normal. Aunque no se haya mencionado antes la primera forma normal, cuando se introdujo el modelo relacional en el Capítulo 3, se afirmó que los valores de los atributos deben ser atómicos.

Algunos tipos de valores no atómicos pueden resultar útiles, aunque deben utilizarse con cuidado. Por ejemplo, los atributos con valores compuestos suelen resultar útiles, y los atributos de tipo conjunto también resultan útiles en muchos casos, que es el motivo por el que el modelo E-R los soporta. En muchos dominios en los que las entidades tienen una estructura compleja, la imposición de la representación en la primera forma normal representa una carga innecesaria para el programador de las aplicaciones, que tiene que escribir código para convertir los datos a su forma atómica. También hay sobrecarga en tiempo de ejecución por la conversión de los datos a la forma atómica y desde ella. Por tanto, el soporte de los valores no atómicos puede resultar muy útil en esos dominios. De hecho, los sistemas modernos de bases de datos soportan muchos tipos de valores no atómicos, como se verá en los capítulos 8 y 9. Sin embargo, en este capítulo nos limitaremos a las relaciones en la primera forma normal.

7.2. DIFICULTADES EN EL DISEÑO DE BASES DE DATOS RELACIONALES

Antes de continuar con el estudio de las formas normales hay que examinar lo que puede salir mal en un mal diseño de bases de datos. Entre las propiedades indeseables que puede tener un mal diseño están:

- Repetición de la información
- Imposibilidad de la representación de determinada información

Estos problemas se estudiarán con ayuda de un diseño de base de datos modificado para el ejemplo bancario: A diferencia del esquema de relación utilizado en los capítulos 3 a 6, supóngase que la información relativa a los préstamos se guarda en una sola relación, *empréstito*, que se define mediante el esquema de relación

Esquema-empréstito = (*nombre-sucursal, ciudad-sucursal, activo, nombre-cliente, número-préstamo, importe*)

La Figura 7.1 muestra un ejemplo de la relación *empréstito* (*esquema-empréstito*). Cada tupla *t* de la relación *empréstito* tiene el siguiente significado intuitivo:

- *t*[*activo*] es el volumen de activo de la sucursal denominada *t*[*nombre-sucursal*].
- *t*[*ciudad-sucursal*] es la ciudad en la que se ubica la sucursal denominada *t*[*nombre-sucursal*].
- *t*[*número-préstamo*] es el número asignado al préstamo concedido por la sucursal denominada *t*[*nombre-sucursal*] al cliente llamado *t*[*nombre-cliente*].
- *t*[*importe*] es el importe del préstamo cuyo número es *t*[*número-préstamo*].

Supóngase que se desea añadir un nuevo préstamo a la base de datos. Digamos que el préstamo se lo concede la sucursal de Navacerrada a la señora Fernández por un importe de 1500 €. Sea el *número-préstamo* P-31.

(Navacerrada, Aluche, 1.700.000, Fernández, P-31, 1.500)

nombre-sucursal	*ciudad-sucursal*	*activo*	*nombre-cliente*	*número-préstamo*	*importe*
Centro	Arganzuela	9.000.000	Santos	P-17	1.000
Moralzarzal	La Granja	2.100.000	Gómez	P-23	2.000
Navacerrada	Aluche	1.700.000	López	P-15	1.500
Centro	Arganzuela	9.000.000	Sotoca	P-14	1.500
Becerril	Aluche	400.000	Santos	P-93	500
Collado Mediano	Aluche	8.000.000	Abril	P-11	900
Navas de la Asunción	Alcalá de Henares	300.000	Valdivieso	P-29	1.200
Segovia	Cerceda	3.700.000	López	P-16	1.300
Centro	Arganzuela	9.000.000	González	P-18	2.000
Navacerrada	Aluche	1.700.000	Rodríguez	P-25	2.500
Galapagar	Arganzuela	7.100.000	Amo	P-10	2.200

FIGURA 7.1. Relación *empréstito* de ejemplo.

En el diseño hace falta una tupla con los valores de todos los atributos de *Esquema-empréstito*. Por tanto, hay que repetir los datos del activo y de la ciudad de la sucursal de Navacerrada y añadir la tupla a la relación *empréstito*. En general, los datos del activo y de la ciudad deben aparecer una vez para cada préstamo concedido por esa sucursal.

La repetición de la información en el diseño alternativo no es deseable. La repetición de la información desaprovecha el espacio. Además, complica la actualización de la base de datos. Supóngase, por ejemplo, que el activo de la sucursal de Navacerrada cambia de 1.700.000 a 1.900.000. Con el diseño original hay que modificar una tupla de la relación *sucursal*. Con el diseño alternativo hay que modificar muchas tuplas de la relación *empréstito*. Por tanto, las actualizaciones resultan más costosas con el diseño alternativo que con el original. Cuando se lleva a cabo la actualización en la base de datos alternativa hay que asegurarse de que se actualicen *todas* las tuplas correspondientes a la sucursal de Navacerrada, o la base de datos mostrará dos valores diferentes del activo para esa sucursal.

Esa observación resulta fundamental para la comprensión del motivo por el que el diseño alternativo es malo. Se sabe que cada sucursal bancaria tiene un valor único del activo, por lo que dado el nombre de una sucursal se puede identificar de manera única el valor del activo. Por otro lado, se sabe que cada sucursal puede conceder muchos préstamos por lo que, dado el nombre de una sucursal, no se puede determinar de manera única el número de un préstamo. En otras palabras, se dice que se cumple la *dependencia funcional*

$$\textit{nombre-sucursal} \rightarrow \textit{activo}$$

para *Esquema-empréstito*, pero no se espera que se cumpla la dependencia funcional *nombre-sucursal* → *número-préstamo*. El hecho de que cada sucursal tenga un valor concreto del activo y el de que cada sucursal conceda préstamos son independientes y, como se ha visto, esos hechos quedan mejor representados en relaciones distintas. Se verá que se pueden utilizar las dependencias funcionales para la especificación formal cuando el diseño de la base de datos es bueno.

Otro problema del diseño *Esquema-empréstito* es que no se puede representar de manera directa la información relativa a cada sucursal (*nombre-sucursal*, *ciudad-sucursal*, *activo*) a menos que haya como mínimo un préstamo en esa sucursal. Esto se debe a que las tuplas de la relación *empréstito* exigen los valores de *número-préstamo*, *importe* y *nombre-cliente*.

Una solución para este problema es introducir los *valores nulos*, como se hizo para tratar las actualizaciones mediante las vistas. Hay que recordar, no obstante, que los valores nulos resultan difíciles de manejar, como se vio en el Apartado 3.3.4. Si no se desea tratar con los valores nulos se puede crear la información sobre cada sucursal sólo después de que se formule la primera solicitud de préstamo en esa sucursal. Lo que es peor aún, habrá que eliminar esa información cuando se hayan pagado todos los préstamos. Evidentemente, esta situación no resulta deseable ya que, con el diseño original de la base de datos, la información sobre la sucursal estaba disponible independientemente de si se mantenía algún préstamo vivo en la sucursal, y sin recurrir a los valores nulos.

7.3. DEPENDENCIAS FUNCIONALES

Las dependencias funcionales desempeñan un papel fundamental en la diferenciación entre los buenos diseños de bases de datos y los malos. Una **dependencia funcional** es un tipo de restricción que constituye una generalización del concepto de *clave*, como se estudió en los capítulos 2 y 3.

7.3.1. Conceptos básicos

Las dependencias funcionales son restricciones del conjunto de relaciones legales. Permiten expresar hechos sobre la empresa que se modela con la base de datos.

En el Capítulo 2 se definió el concepto de *superclave* de la manera siguiente. Sea R el esquema de una relación. El subconjunto K de R es una **superclave** de R si, en cualquier relación legal $r(R)$, para todos los pares t_1 y t_2 de tuplas de r tales que $t_1 \neq t_2$, $t_1 [K] \neq t_2 [K]$. Es decir, ningún par de tuplas de una relación legal $r(R)$ puede tener el mismo valor para el conjunto de atributos K.

El concepto de dependencia funcional generaliza la noción de superclave. Considérese el esquema de una relación R y sean $\alpha \subseteq R$ y $\beta \subseteq R$. La **dependencia funcional**

$$\alpha \rightarrow \beta$$

se cumple para el esquema R si, en cualquier relación legal $r (R)$, para todos los pares de tuplas t_1 y t_2 de r tales que $t_1 [\alpha] = t_2 [\alpha]$, también ocurre que $t_1 [\beta] = t_2 [\beta]$.

Empleando la notación para la dependencia funcional, se dice que K es una superclave de R si $K \rightarrow R$. Es decir, K es una superclave si, siempre que $t_1 [K] = t_2 [K]$, también se produce que $t_1 [R] = t_2 [R]$ (es decir, $t_1 = t_2$).

Las dependencias funcionales nos permiten expresar las restricciones que no se pueden expresar con las superclaves. Considérese el esquema

Esquema-info-préstamo = (*número-préstamo, nombre-sucursal, nombre-cliente, importe*)

que es una simplificación de *Esquema-empréstito*, que se ha visto anteriormente. El conjunto de dependencias funcionales que se espera que se cumplan en este esquema de relación es

número-préstamo → *importe*
número-préstamo → *nombre-sucursal*

Sin embargo, no se espera que se cumpla la dependencia funcional

número-préstamo → *nombre-cliente*

ya que, en general, cada préstamo se puede conceder a más de un cliente (por ejemplo, a los dos integrantes de una pareja marido-esposa).

Las dependencias funcionales se utilizarán de dos maneras:

1. Para probar las relaciones y ver si son legales según un conjunto dado de dependencias funcionales. Si una relación r es legal según el conjunto F de dependencias funcionales, se dice que r **satisface** F.
2. Para especificar las restricciones del conjunto de relaciones legales. Así, *sólo* habrá que preocuparse por las relaciones que satisfagan un conjunto dado de dependencias funcionales. Si uno desea restringirse a las relaciones del esquema R que satisfagan el conjunto F de dependencias funcionales, se dice que F **se cumple** en R.

Considérese la relación r de la Figura 7.2, para ver las dependencias funcionales que se satisfacen. Obsérvese que se satisface $A \rightarrow C$. Hay dos tuplas que tienen un valor para A de a_1. Estas tuplas tienen el mismo valor para C, por ejemplo, c_1. De manera parecida, las dos tuplas con un valor para A de a_2 tienen el mismo valor para C, c_2. No hay más pares de tuplas diferentes que tengan el mismo valor para A. Sin embargo, la dependencia funcional $C \rightarrow A$ no se satisface. Para verlo, considérense las tuplas $t_1 = (a_2, b_3, c_2, d_3)$ y $t_2 = (a_3, b_3, c_2, d_4)$. Estas dos tuplas tienen los mismos valores para C, c_2, pero tienen valores diferentes para A, a_2 y a_3, respectivamente. Por tanto, se ha hallado un par de tuplas t_1 y t_2 tales que $t_1[C] = t_2[C]$, pero $t_1[A] \neq t_2[A]$.

r satisface muchas otras dependencias funcionales, incluida, por ejemplo, la dependencia funcional $AB \rightarrow D$. Obsérvese que se utiliza AB como abreviatura de $\{A,B\}$, para adecuarnos al uso estándar. Obsérvese que no hay ningún par de tuplas diferentes t_1 y t_2 tales que $t_1[AB] = t_2[AB]$. Por tanto, si $t_1[AB] = t_2[AB]$, debe ser que $t_1 = t_2$ y, por tanto, $t_1[D] = t_2[D]$. Así, r satisface $AB \rightarrow D$.

Se dice que algunas dependencias funcionales son **triviales** porque las satisfacen todas las relaciones. Por ejemplo, $A \rightarrow A$ la satisfacen todas las relaciones que impliquen al atributo A. La lectura literal de la definición de dependencia funcional deja ver que, para todas las tuplas t_1 y t_2 tales que $t_1[A] = t_2[A]$, se cumple que $t_1[A] = t_2[A]$. De manera parecida, $AB \rightarrow A$ la satisfacen todas las relaciones que impliquen al atributo A. En general, una dependencia funcional de la forma $\alpha \rightarrow \beta$ es **trivial** si $\beta \subseteq \alpha$.

Para distinguir entre los conceptos de que una relación satisfaga una dependencia y que una dependencia se cumpla en un esquema hay que volver al ejemplo del banco. Si se considera la relación *cliente* (en *Esquema-cliente*) de la Figura 7.3, puede verse que se satisface *calle-cliente* → *ciudad-cliente*. Sin embargo, se sabe que en el mundo real dos ciudades pueden tener calles que se llamen igual. Por tanto, resulta posible, en un momento dado, tener un ejemplar de la relación *cliente* en la que no se satisfaga *calle-cliente* → *ciudad-cliente*. Por consiguiente, no se incluirá *calle-cliente* → *ciudad-cliente* en el conjunto de dependencias funcionales que se cumplen en *Esquema-cliente*.

En la relación *préstamo* (de *Esquema-préstamo*) de la Figura 7.4 se puede ver que se satisface la dependencia *número-préstamo* → *importe*. A diferencia del caso de *ciudad-cliente* y *calle-cliente* de *Esquema-cliente*, se sabe que en la empresa real que se está modelando se exige que cada préstamo tenga un único importe. Por tanto, se desea exigir que la relación *préstamo* satisfaga siempre *número-préstamo* → *importe*. En otras palabras, se exige que la restricción *número-préstamo* → *importe* se cumpla en *Esquema-préstamo*.

En la relación *sucursal* de la Figura 7.5 puede verse que se satisface *nombre-sucursal* → *activo*, igual que ocurre con *activo* → *nombre-sucursal*. Se desea exigir

A	*B*	*C*	*D*
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

FIGURA 7.2. Relación de ejemplo *r*.

nombre-cliente	*calle-cliente*	*ciudad-cliente*
Santos	Mayor	Peguerinos
Gómez	Carretas	Cerceda
López	Mayor	Peguerinos
Pérez	Carretas	Cerceda
Rupérez	Ramblas	León
Abril	Preciados	Valsaín
Valdivieso	Goya	Vigo
Fernández	Jazmín	León
González	Arenal	La Granja
Rodríguez	Yeserías	Cádiz
Amo	Embajadores	Arganzuela
Badorrey	Delicias	Valsaín

FIGURA 7.3. La relación *cliente*.

número-préstamo	*nombre-sucursal*	*importe*
P-17	Centro	1.000
P-23	Moralzarzal	2.000
P-15	Navacerrada	1.500
P-14	Centro	1.500
P-93	Becerril	500
P-11	Collado Mediano	900
P-29	Navas de la Asunción	1.200
P-16	Segovia	1.300
P-18	Centro	2.000
P-25	Navacerrada	2.500
P-10	Galapagar	2.200

FIGURA 7.4. La relación *préstamo*.

nombre-sucursal	*ciudad-sucursal*	*activo*
Centro	Arganzuela	9.000.000
Moralzarzal	La Granja	2.100.000
Navacerrada	Aluche	1.700.000
Becerril	Aluche	400.000
Collado Mediano	Aluche	8.000.000
Navas de la Asunción	Alcalá de Henares	300.000
Segovia	Cerceda	3.700.000
Galapagar	Arganzuela	7.100.000

FIGURA 7.5. La relación *sucursal*.

que se cumpla *nombre-sucursal* → *activo* en *Esquema-sucursal*. Sin embargo, no se desea exigir que se cumpla *activo* → *nombre-sucursal*, ya que es posible tener varias sucursales con el mismo valor del activo.

En lo que viene a continuación se da por supuesto que, cuando se diseña una base de datos relacional, se enumeran en primer lugar las dependencias funcionales que se deben cumplir siempre. En el ejemplo del banco, en la lista de dependencias figuran:

- En *Esquema-sucursal*:
  nombre-sucursal → *ciudad-sucursal*
  nombre-sucursal → *activo*
- En *Esquema-cliente*:
  nombre-cliente → *ciudad-cliente*
  nombre-cliente → *calle-cliente*
- En *Esquema-préstamo*:
  número-préstamo → *importe*
  número-préstamo → *nombre-sucursal*
- En *Esquema-prestatario*:
  Ninguna dependencia funcional
- En *Esquema-cuenta*:
  número-cuenta → *nombre-sucursal*
  número-cuenta → *saldo*
- En *Esquema-impositor*:
  Ninguna dependencia funcional

7.3.2. Cierre de un conjunto de dependencias funcionales

No es suficiente considerar el conjunto dado de dependencias funcionales. También hay que considerar *todas* las dependencias funcionales que se cumplen. Se verá que, dado un conjunto F de dependencias funcionales, se puede probar que se cumplen otras dependencias funcionales determinadas. Se dice que esas dependencias funcionales están «implicadas lógicamente» por F.

De manera más formal, dado un esquema relacional R, una dependencia funcional f de R está **implicada lógicamente** por un conjunto de dependencias funcionales F de R si cada ejemplar de la relación $r(R)$ que satisface F satisface también f.

Supóngase que se tiene un esquema de relación $R = (A, B, C, G, H, I)$ y el conjunto de dependencias funcionales

$$A \rightarrow B$$
$$A \rightarrow C$$
$$CG \rightarrow H$$
$$CG \rightarrow I$$
$$B \rightarrow H$$

La dependencia funcional

$$A \rightarrow H$$

está implicada lógicamente. Es decir, se puede demostrar que, siempre que el conjunto dado de dependencias funcionales se cumple en una relación, en la relación también se debe cumplir $A \rightarrow H$. Supóngase que t_1 y t_2 son tuplas tales que

$$t_1 [A] = t_2 [A]$$

Como se tiene que $A \rightarrow B$, se deduce de la definición de dependencia funcional que

$$t_1 [B] = t_2 [B]$$

Entonces, como se tiene que $B \rightarrow H$, se deduce de la definición de dependencia funcional que

$$t_1 [H] = t_2 [H]$$

Por tanto, se ha demostrado que, siempre que t_1 y t_2 sean tuplas tales que $t_1 [A] = t_2 [A]$, debe ocurrir que $t_1 [H] = t_2 [H]$. Pero ésa es exactamente la definición de $A \rightarrow H$.

Sea F un conjunto de dependencias funcionales. El **cierre** de F, denotado por F^+, es el conjunto de todas las dependencias funcionales implicadas lógicamente en F. Dado F, se puede calcular F^+ directamente a partir de la definición formal de dependencia funcional. Si F fuera de gran tamaño, este proceso sería prolongado y difícil. Este cálculo de F^+ requiere argumentos del tipo que se acaba de utilizar para demostrar que $A \rightarrow H$ está en el cierre del conjunto de ejemplo de dependencias.

Los **axiomas**, o reglas de inferencia, proporcionan una técnica más sencilla para el razonamiento sobre las depen-

dencias funcionales. En las reglas que se ofrecen a continuación se utilizan las letras griegas (α, β, γ, . . .) para los conjuntos de atributos y las letras latinas mayúsculas desde el comienzo del alfabeto para los atributos individuales. Se utiliza ab para denotar $\alpha \cup \beta$.

Se pueden utilizar las tres reglas siguientes para hallar las dependencias funcionales implicadas lógicamente. Aplicando estas reglas *repetidamente*, se puede hallar todo F^+, dado F. Este conjunto de reglas se denomina **axiomas de Armstrong** en honor de la persona que las propuso por primera vez.

- **Regla de la reflexividad.** Si α es un conjunto de atributos y $\beta \subseteq \alpha$, entonces se cumple que $\alpha \rightarrow \beta$.
- **Regla de la aumentatividad.** Si se cumple que $\alpha \rightarrow \beta$ y γ es un conjunto de atributos, entonces se cumple que $\gamma\alpha \rightarrow \gamma\beta$.
- **Regla de la transitividad.** Si se cumple que $\alpha \rightarrow \beta$ y también se cumple que $\beta \rightarrow \gamma$, entonces se cumple que $\alpha \rightarrow \gamma$.

Los axiomas de Armstrong son **correctos** porque no generan dependencias funcionales incorrectas. Son **completos**, porque, para un conjunto dado F de dependencias funcionales, permiten generar todo F^+. Las notas bibliográficas proporcionan referencias de las pruebas de su corrección y de su completitud.

Aunque los axiomas de Armstrong son completos, resulta difícil utilizarlos directamente para el cálculo de F^+. Para simplificar más las cosas se relacionan unas reglas adicionales. Resulta posible utilizar los axiomas de Armstrong para probar que estas reglas son correctas (véanse los ejercicios 7.8, 7.9 y 7.10).

- **Regla de la unión.** Si se cumple que $\alpha \rightarrow \beta$ y que $\alpha \rightarrow \gamma$, entonces se cumple que $\alpha \rightarrow \beta\gamma$.
- **Regla de la descomposición.** Si se cumple que $\alpha \rightarrow \beta\gamma$, entonces se cumple que $\alpha \rightarrow \beta$ y que $\alpha \rightarrow \gamma$.
- **Regla de la pseudotransitividad.** Si se cumple que $\alpha \rightarrow \beta$ y que $\gamma\beta \rightarrow \delta$, entonces se cumple que $\alpha\gamma \rightarrow \delta$.

Apliquemos ahora las reglas al ejemplo del esquema $R = (A, B, C, G, H, I)$ y el conjunto F de dependencias funcionales $\{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$. A continuación se relacionan varios miembros de F^+:

- $A \rightarrow H$. Dado que se cumplen $A \rightarrow B$ y $B \rightarrow H$, se aplica la regla de transitividad. Obsérvese que resultaba mucho más sencillo emplear los axiomas de Armstrong para demostrar que se cumple que $A \rightarrow H$ que deducirlo directamente a partir de las definiciones, como se ha hecho anteriormente en este apartado.
- $CG \rightarrow HI$. Dado que $CG \rightarrow H$ y $CG \rightarrow I$, la regla de unión implica que $CG \rightarrow HI$.
- $AG \rightarrow I$. Dado que $A \rightarrow C$ y $CG \rightarrow I$, la regla de pseudotransitividad implica que se cumple que $AG \rightarrow I$.

Otra manera de hallar que se cumple que $AG \rightarrow I$ es la siguiente. Se utiliza la regla de aumentatividad en $A \rightarrow C$ para inferir que $AG \rightarrow CG$. Aplicando la regla de transitividad a esta dependencia y $CG \rightarrow I$, se infiere que $AG \rightarrow I$.

La Figura 7.6 muestra un procedimiento que demuestra formalmente el modo de utilizar los axiomas de Armstrong para calcular F^+. En este procedimiento, cuando se añade una dependencia funcional a F^+, puede que ya esté presente y, en ese caso, no hay ninguna modificación en F^+. También se verá una manera alternativa de calcular F^+ en el Apartado 7.3.3.

Los términos a la derecha y a la izquierda de una dependencia funcional son subconjuntos de R. Dado que un conjunto de tamaño n tiene 2^n subconjuntos, hay un total de $2 \times 2^n = 2^{n+1}$ dependencias funcionales posibles, donde n es el número de atributos de R. Cada iteración del bucle repeat del procedimiento, salvo la última, añade como mínimo una dependencia funcional a F^+. Por tanto, se garantiza que el procedimiento se termine.

7.3.3. Cierre de un conjunto de atributos

Para comprobar si un conjunto α es una superclave hay que diseñar un algoritmo para el cálculo del conjunto de atributos determinados funcionalmente por α. Una manera de hacerlo es calcular F^+, tomar todas las dependencias funcionales con α como término de la izquierda y tomar la unión de los términos de la derecha de todas esas dependencias. Sin embargo, hacer esto puede resultar costoso, ya que F^+ puede ser de gran tamaño.

El algoritmo eficiente para calcular el conjunto de atributos determinados funcionalmente por a no sólo resulta útil para comprobar si a es una superclave, sino también para otras tareas, como se verá más adelante en este apartado.

Sea α un conjunto de atributos. Al conjunto de todos los atributos determinados funcionalmente por α bajo un conjunto F de dependencias funcionales se le denomina **cierre** de α bajo F; se denota mediante α^+. La Figura 7.7 muestra un algoritmo, escrito en pseudocó-

```
F+ = F
repeat
  for each dependencia funcional f de F+
    aplicar las reglas de reflexividad y de aumentatividad a f
    añadir las dependencias funcionales resultantes a F+
  for each pareja de dependencias funcionales f1 y f2 de F+
    if f1 y f2 pueden combinarse mediante la transitividad
      añadir la dependencia funcional resultante a F+
until F+ no cambie más
```

FIGURA 7.6. Procedimiento para calcular F^+.

```
resultado := α;
while (cambios en resultado) do
    for each dependencia funcional β → γ in F do
      begin
        if β ⊆ resultado then resultado := resultado ∪ γ;
      end
```

FIGURA 7.7. Algoritmo para el cálculo de α^+, el cierre de α bajo *F*.

digo, para calcular α^+. La entrada es un conjunto F de dependencias funcionales y el conjunto α de atributos. La salida se almacena en la variable *resultado*.

Para ilustrar el modo en que trabaja el algoritmo se utilizará para calcular $(AG)^+$ con las dependencias funcionales definidas en el Apartado 7.3.2. Se comienza con *resultado* = AG. La primera vez que se ejecuta el bucle **while** para comprobar cada dependencia funcional se halla que

- $A \rightarrow B$ hace que se incluya B en *resultado*. Para ver este hecho, se observa que $A \rightarrow B$ se halla en F y $A \subseteq$ *resultado* (que es AG), por lo que *resultado* := *resultado* $\cup B$.
- $A \rightarrow C$ hace que *resultado* se transforme en $ABCG$.
- $CG \rightarrow H$ hace que *resultado* se transforme en $ABCGH$.
- $CG \rightarrow I$ hace que *resultado* se transforme en $ABCGHI$.

La segunda vez que se ejecuta el bucle **while** no se añaden atributos nuevos a *resultado*, y se termina el algoritmo.

Veamos ahora el motivo por el que el algoritmo de la Figura 7.7 es correcto. El primer paso es correcto, ya que $\alpha \rightarrow \alpha$ se cumple siempre (por la regla de reflexividad). Se asegura que, para cualquier subconjunto β de *resultado*, $\alpha \rightarrow \beta$. Dado que se inicia el bucle **while** con $\alpha \rightarrow$ *resultado* como cierto, sólo se puede añadir γ a *resultado* si $\beta \subseteq$ *resultado* y $\beta \rightarrow \gamma$. Pero, entonces, *resultado* $\rightarrow \beta$ por la regla de reflexividad, por lo que $\alpha \rightarrow \beta$ por transitividad. Otra aplicación de la transitividad demuestra que $\alpha \rightarrow \gamma$ (utilizando $\alpha \rightarrow \beta$ y $\beta \rightarrow \gamma$). La regla de la unión implica que $\alpha \rightarrow$ *resultado* $\cup\ \gamma$, por lo que α determina funcionalmente cualquier resultado nuevo generado en el bucle **while**. Por tanto, cualquier atributo devuelto por el algoritmo se halla en α^+.

Resulta sencillo ver que el algoritmo halla todo α^+. Si hay un atributo de α^+ que no se halle todavía en *resultado*, debe haber una dependencia funcional $\beta \rightarrow \gamma$ para la que $\beta \subseteq$ *resultado* y, como mínimo, un atributo de γ no se halla en *resultado*.

Resulta que, en el peor de los casos, este algoritmo puede tardar un tiempo proporcional al cuadrado del tamaño de F. Hay un algoritmo más rápido (aunque ligeramente más complejo) que se ejecuta en un tiempo proporcional al tamaño de F; este algoritmo se presenta como parte del Ejercicio 7.14.

Hay varios usos para el algoritmo de cierre de atributos:

- Comprobar si α es una superclave, se calcula α^+ y se comprueba si α^+ contiene todos los atributos de R.
- Se puede comprobar si se cumple la dependencia funcional $\alpha \rightarrow \beta$ (o, en otras palabras, si se halla en F^+), comprobando si $\beta \subseteq \alpha^+$. Es decir, se calcula α^+ empleando el cierre de los atributos y luego se comprueba si contiene a β. Esta prueba resulta especialmente útil, como se verá más adelante en este capítulo.
- Ofrece una manera alternativa de calcular F^+: para cada $\gamma \subseteq R$ se halla el cierre γ^+ y, para cada $S \subseteq \gamma^+$, se genera una dependencia funcional $\gamma \rightarrow S$.

7.3.4. Recubrimiento canónico

Supóngase que se tiene un conjunto de dependencias funcionales F de un esquema de relación. Siempre que un usuario lleve a cabo una actualización de la relación el sistema de bases de datos debe asegurarse de que la actualización no viole ninguna dependencia funcional, es decir, que se satisfagan todas las dependencias funcionales de F en el nuevo estado de la base de datos.

El sistema debe retroceder la actualización si viola alguna dependencia funcional del conjunto F.

Se puede reducir el esfuerzo empleado en la comprobación de las violaciones comprobando un conjunto simplificado de dependencias funcionales que tenga el mismo cierre que el conjunto dado. Cualquier base de datos que satisfaga el conjunto simplificado de dependencias funcionales satisfará también el conjunto original y viceversa, ya que los dos conjuntos tienen el mismo cierre. Sin embargo, el conjunto simplificado resulta más sencillo de comprobar. En breve se verá el modo en que se puede crear el conjunto simplificado. Antes, hacen falta algunas definiciones.

Se dice que un atributo de una dependencia funcional es **raro** si se puede eliminar sin modificar el cierre del conjunto de dependencias funcionales. La definición formal de los **atributos raros** es la siguiente. Considérese un conjunto F de dependencias funcionales y la dependencia funcional $\alpha \rightarrow \beta$ de F.

- El atributo A es raro en α si $A \in \alpha$ y F implica lógicamente a $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
- El atributo A es raro en β si $A \in \beta$ y el conjunto de dependencias funcionales $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ implica lógicamente a F.

Por ejemplo, supóngase que se tienen las dependencias funcionales $AB \rightarrow C$ y $A \rightarrow C$ de F. Entonces, B es raro en $AB \rightarrow C$. Otro ejemplo más: supóngase que se tienen las dependencias funcionales $AB \rightarrow CD$ y $A \rightarrow C$ de F. Entonces, C será raro en el lado derecho de $AB \rightarrow CD$.

Hay que prestar atención a la dirección de las implicaciones cuando se utiliza la definición de los atributos raros: si se intercambian el lado derecho y el izquierdo la implicación se cumplirá *siempre*. Es decir, $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$ siempre implica lógicamente a F, y también F implica siempre lógicamente a $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$

He aquí el modo de comprobar de manera eficiente si un atributo es raro. Sea R el esquema de la relación y F el conjunto dado de dependencias funcionales que se cumplen en R.

Considérese el atributo A de la dependencia $\alpha \rightarrow \beta$.

- Si $A \in \beta$, para comprobar si A es raro hay que considerar el conjunto
$$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$$
y comprobar si $\alpha \rightarrow A$ puede inferirse a partir de F'. Para ello hay que calcular α^+ (el cierre de α) bajo F'; si α^+ incluye a A, entonces A es raro en β.
- Si $A \in \alpha$, para comprobar si A es raro, sea $\gamma = \alpha - \{A\}$, hay que comprobar si se puede inferir que $\gamma \rightarrow \beta$ a partir de F. Para ello hay que calcular γ^+ (el cierre de γ) bajo F; si γ^+ incluye todos los atributos de β, entonces A es raro en α.

Por ejemplo, supóngase que F contiene $AB \rightarrow CD$, $A \rightarrow E$ y $E \rightarrow C$. Para comprobar si C es raro en $AB \rightarrow CD$, hay que calcular el cierre de los atributos de AB bajo $F' = \{AB \rightarrow D, A \rightarrow E$ y $E \rightarrow C\}$. El cierre es $ABCDE$, que incluye a CD, por lo que se infiere que C es raro.

El **recubrimiento canónico** F_c de F es un conjunto de dependencias tales que F implica lógicamente todas las dependencias de F_c y F_c implica lógicamente todas las dependencias de F. Además, F_c debe tener las propiedades siguientes:

- Ninguna dependencia funcional de F_c contiene atributos raros.
- El lado izquierdo de cada dependencia funcional de F_c es único. Es decir, no hay dos dependencias $\alpha_1 \rightarrow \beta_1$ y $\alpha_2 \rightarrow \beta_2$ de F_c tales que $\alpha_1 = \alpha_2$.

El recubrimiento canónico del conjunto de dependencias funcionales F puede calcularse como se muestra en la Figura 7.8. Es importante destacar que, cuando se comprueba si un atributo es raro, la comprobación utiliza las dependencias del valor actual de F_c, y **no** las dependencias de F. Si una dependencia funcional sólo contiene un atributo en su lado derecho, por ejemplo, $A \rightarrow C$, y se descubre que ese atributo es raro, se obtiene una dependencia funcional con el lado derecho vacío. Hay que eliminar estas dependencias funcionales.

Se puede demostrar que el recubrimiento canónico de F, F_c, tiene el mismo cierre que F; por tanto, la comprobación de si se satisface F_c es equivalente a la comprobación de si se satisface F. Sin embargo, F_c es mínimo en un cierto sentido: no contiene atributos raros, y combina las dependencias funcionales con el lado izquierdo idéntico. Resulta más económico comprobar F_c que comprobar el propio F.

Considérese el siguiente conjunto F de dependencias funcionales para el esquema (A,B,C):

$$A \rightarrow BC$$
$$B \rightarrow C$$
$$A \rightarrow B$$
$$AB \rightarrow C$$

Calcúlese el recubrimiento canónico de F.

- Hay dos dependencias funcionales con el mismo conjunto de atributos a la izquierda de la flecha:
$$A \rightarrow BC$$
$$A \rightarrow B$$
Estas dependencias funcionales se combinan en $A \rightarrow BC$.
- A es raro en $AB \rightarrow C$ porque F implica lógicamente a $(F - \{AB \rightarrow C\}) \cup \{B \rightarrow C\}$. Esta aseveración es cierta porque $B \rightarrow C$ ya se halla en el conjunto de dependencias funcionales.
- C es raro en $A \rightarrow BC$, ya que $A \rightarrow BC$ está implicada lógicamente por $A \rightarrow B$ y $B \rightarrow C$.

Por tanto, el recubrimiento canónico es:

$$A \rightarrow B$$
$$B \rightarrow C$$

Dado un conjunto F de dependencias funcionales, puede que toda una dependencia funcional del conjunto sea rara, en el sentido de que su eliminación no modifica el cierre de F. Se puede demostrar que el recubrimiento canónico F_c de F no contiene esa dependencia funcional rara. Supóngase que, por el contra-

```
Fc = F
repeat
    Utilizar la regla de unión para sustituir las dependencias de Fc de la forma
                α1 → β1 y α1 → β2 con α1 → β1 β2.
    Hallar una dependencia funcional α → β de Fc con un atributo raro en α o en β.
        /* Nota: la comprobación de los atributos raros se lleva a cabo empleando Fc, no F */
    Si se halla algún atributo raro, hay que eliminarlo de α → β.
until Fc ya no cambie.
```

FIGURA 7.8. Cálculo del recubrimiento canónico.

rio, existiera esa dependencia rara en F_c. Los atributos del lado derecho de la dependencia serían raros, lo que no es posible por la definición de los recubrimientos canónicos.

Puede que el recubrimiento canónico no sea único. Por ejemplo, considérese el conjunto de dependencias funcionales $F = \{A \rightarrow BC, B \rightarrow AC$ y $C \rightarrow AB\}$. Si se aplica la prueba de rareza a $A \rightarrow BC$ se descubre que tanto B como C son raros bajo F. Sin embargo, sería incorrecto eliminarlos a los dos. El algoritmo para hallar el recubrimiento canónico selecciona uno de ellos y lo elimina. Entonces,

1. Si se elimina C, se obtiene el conjunto $F' = \{A \rightarrow B, B \rightarrow AC$ y $C \rightarrow AB\}$. Ahora B ya no es raro en el lado derecho de $A \rightarrow B$ bajo F'. Siguiendo con el algoritmo se descubre que A y B son raros en el lado derecho de $C \rightarrow AB$, lo que genera dos recubrimientos canónicos,
   $F_c = \{A \rightarrow B, B \rightarrow C$ y $C \rightarrow A\}$ y
   $F_c = \{A \rightarrow B, B \rightarrow AC$ y $C \rightarrow B\}$.
2. Si se elimina B, se obtiene el conjunto $\{A \rightarrow C, B \rightarrow AC$ y $C \rightarrow AB\}$. Este caso es simétrico al anterior, y genera dos recubrimientos canónicos,
   $F_c = \{A \rightarrow C, C \rightarrow B$ y $B \rightarrow A\}$ y
   $F_c = \{A \rightarrow C, B \rightarrow C$ y $C \rightarrow AB\}$.

Como ejercicio el lector debe intentar hallar otro recubrimiento canónico de F.

7.4. DESCOMPOSICIÓN

El mal diseño del Apartado 7.2 sugiere que se deben *descomponer* los esquemas de relación que tienen muchos atributos en varios esquemas con menos atributos. Una descomposición poco cuidadosa, no obstante, puede llevar a otra modalidad de mal diseño.

Considérese un diseño alternativo en el que se descompone *Esquema-empréstito* en los dos esquemas siguientes:

Esquema-sucursal-cliente = (*nombre-sucursal, ciudad-sucursal, activo, nombre-cliente*)
Esquema-cliente-préstamo = (*nombre-cliente, número-préstamo, importe*)

Empleando la relación *empréstito* de la Figura 7.1 se crean las relaciones nuevas *sucursal-cliente* (*Esquema-sucursal-cliente*) y *cliente-préstamo* (*Esquema-cliente-préstamo*):

$$\textit{sucursal-cliente} = \Pi_{\textit{nombre-sucursal, ciudad-sucursal, activo, nombre-cliente}} (\textit{empréstito})$$
$$\textit{cliente-préstamo} = \Pi_{\textit{nombre-cliente, número-préstamo, importe}} (\textit{empréstito})$$

Las Figuras 7.9 y 7.10, respectivamente, muestran las relaciones *sucursal-cliente* y *nombre-cliente* resultantes.

Por supuesto, hay casos en los que hace falta reconstruir la relación *préstamo*. Por ejemplo, supóngase que se desea hallar todas las sucursales que tienen préstamos con importes inferiores a 1000 €. Ninguna relación de la base de datos alternativa contiene esos datos. Hay que reconstruir la relación *empréstito*. Parece que se puede hacer escribiendo

$$\textit{sucursal-cliente} \bowtie \textit{cliente-préstamo}$$

La Figura 7.11 muestra el resultado de calcular *sucursal-cliente* ⋈ *préstamo-cliente*. Cuando se compara esta relación con la relación *empréstito* con la cual comenzamos (Figura 7.1) se aprecia una diferencia: aunque cada tupla que aparece en *empréstito* aparece también en *sucursal-cliente* ⋈ *préstamo-cliente*, hay tuplas de *sucursal-cliente* ⋈ *préstamo-cliente* que no

nombre-sucursal	*ciudad-sucursal*	*activo*	*nombre-cliente*
Centro	Arganzuela	9.000.000	Santos
Moralzarzal	La Granja	2.100.000	Gómez
Navacerrada	Aluche	1.700.000	López
Centro	Arganzuela	9.000.000	Sotoca
Becerril	Aluche	400.000	Santos
Collado Mediano	Aluche	8.000.000	Abril
Navas de la Asunción	Alcalá de Henares	300.000	Valdivieso
Segovia	Cerceda	3.700.000	López
Centro	Arganzuela	9.000.000	González
Navacerrada	Aluche	1.700.000	Rodríguez
Galapagar	Arganzuela	7.100.000	Amo

FIGURA 7.9. La relación *sucursal-cliente*.

nombre-cliente	*número-préstamo*	*importe*
Santos	P-17	1.000
Gómez	P-23	2.000
López	P-15	1.500
Sotoca	P-14	1.500
Santos	P-93	500
Abril	P-11	900
Valdivieso	P-29	1.200
López	P-16	1.300
González	P-18	2.000
Rodríguez	P-25	2.500
Amo	P-10	2.200

FIGURA 7.10. La relación *cliente-préstamo*.

nombre-sucursal	*ciudad-sucursal*	*activo*	*nombre-cliente*	*número-préstamo*	*importe*
Centro	Arganzuela	9.000.000	Santos	P-17	1.000
Centro	Arganzuela	9.000.000	Santos	P-93	500
Moralzarzal	La Granja	2.100.000	Gómez	P-23	2.000
Navacerrada	Aluche	1.700.000	López	P-15	1.500
Navacerrada	Aluche	1.700.000	López	P-16	1.300
Centro	Arganzuela	9.000.000	Sotoca	P-14	1.500
Becerril	Aluche	400.000	Santos	P-17	1.000
Becerril	Aluche	400.000	Santos	P-93	500
Collado Mediano	Aluche	8.000.000	Abril	P-11	900
Navas de la Asunción	Alcalá de Henares	300.000	Valdivieso	P-29	1.200
Segovia	Cerceda	3.700.000	López	P-15	1.500
Segovia	Cerceda	3.700.000	López	P-16	1.300
Centro	Arganzuela	9.000.000	González	P-18	2.000
Navacerrada	Aluche	1.700.000	Rodríguez	P-25	2.500
Galapagar	Arganzuela	7.100.000	Amo	P-10	2.200

FIGURA 7.11. La relación *sucursal-cliente* ⋈ *cliente-préstamo*.

están en *empréstito*. En el ejemplo, *sucursal-cliente* ⋈ *préstamo-cliente* tiene las siguientes tuplas adicionales:

(Centro, Arganzuela, 9.000.000, Santos, P-93, 500)
(Navacerrada, Aluche, 1.700.000, López, P-16, 1.300)
(Becerril, Aluche, 400.000, Santos, P-17, 1.000)
(Segovia, Cerceda, 3.700.000, López, P-15, 1.500)

Considérese la consulta «Hallar todas las sucursales que han concedido un préstamo por un importe inferior a 1.500 €». Si se vuelve a la Figura 7.1 se observa que las únicas sucursales con créditos con importe inferior a 1.500 € son Becerril y Collado Mediano. Sin embargo, al aplicar la expresión

$$\Pi_{nombre\text{-}sucursal}\,(\sigma_{importe\,<\,1.000}\,(sucursal\text{-}cliente \bowtie préstamo\text{-}cliente))$$

obtenemos los nombres de tres sucursales: Becerril, Collado Mediano y Centro.

Un examen más detenido de este ejemplo muestra el motivo. Si resulta que un cliente tiene varios préstamos de diferentes sucursales, no se puede decir el préstamo que pertenece a cada sucursal. Por lo tanto, al reunir *sucursal-cliente* y *cliente-préstamo*, no sólo se obtienen las tuplas que se tenía originariamente en *empréstito*, sino también varias tuplas adicionales. Aunque se tengan *más* tuplas en *sucursal-cliente* ⋈ *cliente-préstamo*, realmente se tiene *menos* información. Ya no es posible, en general, representar en la base de datos la información de los clientes que tienen concedidos préstamos en cada sucursal. Debido a esta pérdida de información se dice que la descomposición de *Esquema-empréstito* en *Esquema-sucursal-cliente* y *Esquema-cliente-préstamo* es una **descomposición con pérdida**, o una **descomposición de reunión con pérdida**. Una descomposición que no es una descomposición con pérdida es una **descomposición de reunión sin pérdida**. Queda claro con este ejemplo que una descomposición de reunión con pérdida supone, en general, un mal diseño de base de datos.

Es interesante averiguar el motivo por el que la descomposición es una descomposición con pérdida. Hay un atributo en común entre *Esquema-cliente-sucursal* y *Esquema-cliente-préstamo*:

Esquema-sucursal-cliente ∩ *Esquema-cliente-préstamo* = {*nombre-cliente*}

El único modo de que se pueda representarse una relación entre, por ejemplo, *número-préstamo* y *nombre-sucursal* es mediante *nombre-cliente*. Esta representación no resulta adecuada porque puede que un cliente tenga concedidos varios préstamos, pero esos préstamos no tienen que haberse obtenido necesariamente de la misma sucursal.

Considérese otro diseño alternativo en el que se descompone *Esquema-empréstito* en los dos esquemas siguientes:

Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activo*)
Esquema-info-préstamo = (*nombre-sucursal, nombre-cliente, número-préstamo, importe*)

Hay un atributo en común entre estos dos esquemas:

Esquema-sucursal-préstamo ∩ *Esquema-cliente-préstamo* = {*nombre-sucursal*}

Por lo tanto, el único modo de poder representar una relación entre, por ejemplo, *nombre-cliente* y *activo* es

mediante *nombre-sucursal*. La diferencia entre este ejemplo y el anterior es que el activo de una sucursal es el mismo, independientemente del cliente al que se haga referencia, mientras que la sucursal prestamista *sí* depende del cliente al que se haga referencia. Para un valor dado de *nombre-sucursal* dado, hay exactamente un valor de *activo* y un valor de *ciudad-sucursal*, mientras que no se puede hacer una afirmación parecida para *nombre-cliente*. Es decir, se cumple la dependencia funcional:

nombre-sucursal → *activo ciudad-sucursal*

pero *nombre-cliente* no determina funcionalmente a *número-préstamo*.

El concepto de reunión sin pérdida resulta fundamental para gran parte del diseño de bases de datos relacionales. Por lo tanto, se volverán a formular los ejemplos anteriores de manera más concisa y formal. Sea R un esquema de relación. Un conjunto de esquemas de relación $\{R_1, R_2, ..., R_n\}$ es una **descomposición** de R si

$$R = R_1 \cup R_2 \cup \ldots \cup R_n$$

Es decir, $\{R_1, R_2, ..., R_n\}$ es una descomposición de R si, para $i = 1, 2,..., n$, cada R_i es un subconjunto de R y cada atributo de R aparece en al menos un R_i.

Sea r una relación del esquema R y $r_i = \prod_{R_i}(r)$ para $i = 1, 2,..., n$. Es decir, $\{r_1, r_2,..., r_n\}$ es la base de datos que resulta de descomponer R en $\{R_1, R_2, ..., R_n\}$. Siempre se cumple que

$$r \subseteq r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

Para comprobar que esta afirmación es cierta considérese una tupla t de la relación r. Cuando se calculan las relaciones $r_1, r_2, \ldots, r_n$, la tupla t da lugar a una tupla t_i en cada r_i, $i = 1, 2, \ldots, n$. Estas n tuplas se combinan para regenerar t cuando se calcula $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$. Los detalles se dejan para completarlos como ejercicio. Por tanto, cada tupla de r aparece en $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$.

En general, $r \neq r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$. Para mostrarlo consideremos el ejemplo anterior, en el que

- $n = 2$.
- R = *Esquema-empréstito*.
- R_1 = *Esquema-sucursal-cliente*.
- R_2 = *Esquema-cliente-préstamo*
- r = la relación mostrada en la Figura 7.1.
- r_1 = la relación mostrada en la Figura 7.9.
- r_2 = la relación mostrada en la Figura 7.10.
- $r_1 \bowtie r_2$ = la relación mostrada en la Figura 7.11.

Obsérvese que las relaciones de las Figuras 7.1 y 7.11 no son iguales.

Para tener una descomposición de reunión sin pérdida hay que imponer restricciones en el conjunto de las relaciones posibles. Se descubrió que la descomposición de *Esquema-empréstito* en *Esquema-sucursal* y *Esquema-info-préstamo* era sin pérdida porque se cumple la dependencia funcional

nombre-sucursal → *ciudad-sucursal activo*

en *Esquema-sucursal*.

Más adelante en este capítulo se introducirán otras restricciones distintas de las dependencias funcionales. Se dice que una relación es **legal** si satisface todas las reglas, o restricciones, que se hayan impuesto en la base de datos.

Sea C un conjunto de restricciones de la base de datos y R un esquema de relación. Una descomposición $\{R_1, R_2, \ldots, R_n\}$ de R es una **descomposición de reunión sin pérdida** si, para todas las relaciones r del esquema R que son legales bajo C

$$r = \prod_{R_1}(r) \bowtie \prod_{R_2}(r) \bowtie \ldots \bowtie \prod_{R_n}(r)$$

En los siguientes apartados se mostrará el modo de comprobar si una descomposición es una descomposición de reunión sin pérdida. La parte principal de este capítulo trata del problema de la especificación de restricciones para las bases de datos y del modo de obtener descomposiciones de reunión sin pérdida que eviten los inconvenientes representados en los ejemplos de malos diseños de bases de datos que se han visto en este apartado.

7.5. PROPIEDADES DESEABLES DE LA DESCOMPOSICIÓN

Se puede utilizar un conjunto dado de dependencias funcionales para diseñar una base de datos relacional en la que no se halle presente la mayor parte de las propiedades no deseables estudiadas en el Apartado 7.2. Cuando se diseñan estos sistemas puede hacerse necesaria la descomposición de una relación en varias relaciones de menor tamaño.

En este apartado se describen las propiedades deseables de las descomposiciones de los esquemas relacionales. En apartados posteriores se describen maneras concretas de descomponer un esquema relacional para obtener las propiedades deseadas. Estos conceptos se ilustran con el esquema *Esquema-empréstito* del Apartado 7.2:

Esquema-empréstito = (*nombre-sucursal, ciudad-sucursal, activo, nombre-cliente, número-préstamo, importe*)

El conjunto F de dependencias funcionales que se exige que se cumplan en *Esquema-empréstito* es

nombre-sucursal → *ciudad-sucursal activo*
número-préstamo → *importe nombre-sucursal*

Como se estudió en el Apartado 7.2, *Esquema-empréstito* es un ejemplo de un mal diseño de base de datos. Supóngase que se descompone en las tres relaciones siguientes:

Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activo*)
Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
Esquema-prestatario = (*nombre-cliente, número-préstamo*)

Puede afirmarse que esta descomposición tiene varias propiedades deseables que se estudiarán a continuación. Obsérvese que estos tres esquemas de relación son precisamente los que se utilizaron anteriormente en los capítulos 3 a 5.

7.5.1. Descomposición de reunión sin pérdida

En el Apartado 7.4 se arguyó que, al descomponer una relación en varias relaciones de menor tamaño, resulta fundamental que la descomposición sea una descomposición sin pérdida. Se puede afirmar que la descomposición del Apartado 7.5 es realmente una descomposición sin pérdida. Para demostrar esta afirmación antes hay que presentar un criterio para determinar si una descomposición es una descomposición con pérdida.

Sea R un esquema de relación, y sea F un conjunto de dependencias funcionales de R. R_1 y R_2 forman una descomposición de R. Esta descomposición es una descomposición de reunión sin pérdida de R si al menos una de las siguientes dependencias se halla en F^+:

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

En otras palabras, si $R_1 \cap R_2$ forma una superclave de R_1 o de R_2, la descomposición de R es una descomposición de reunión sin pérdida. Se puede utilizar el cierre de los atributos para comprobar de manera eficiente la existencia de superclaves, como ya se ha visto.

Ahora se demostrará que la descomposición de *Esquema-empréstito* es una descomposición de reunión sin pérdida mostrando una secuencia de pasos que generan la descomposición. Para empezar se descompone *Esquema-empréstito* en dos esquemas:

Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activo*)
Esquema-info-préstamo = (*nombre-sucursal, nombre-cliente, número-préstamo, importe*)

Dado que *nombre-sucursal* → *ciudad-sucursal activo*, la regla de la aumentatividad para las dependencias funcionales (Apartado 7.3.2) implica que

nombre-sucursal → *nombre-sucursal ciudad-sucursal activo*

Como *Esquema-sucursal* ∩ *Esquema-info-préstamo* = {*nombre-sucursal*}, se concluye que la descomposición inicial es una descomposición de reunión sin pérdida.

A continuación se descompone *Esquema-info-préstamo* en

Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
Esquema-prestatario = (*nombre-cliente, número-préstamo*)

Este paso da lugar a una descomposición de reunión sin pérdida ya que *número-préstamo* es un atributo común y *número-préstamo* → *importe nombre-sucursal*.

En el caso general de la descomposición simultánea de una relación en varias partes, la búsqueda de la descomposición de reunión sin pérdida resulta más complicada. Véanse las notas bibliográficas para encontrar referencias sobre este tema.

Aunque la prueba de la descomposición binaria es, evidentemente, una condición suficiente para la reunión sin pérdida, sólo constituye una condición necesaria si todas las restricciones son dependencias funcionales. Más adelante se verán otros tipos de restricciones (en especial, un tipo de restricción denominado dependencia multivalorada), que pueden asegurar que una descomposición es una reunión sin pérdida aunque no haya ninguna dependencia funcional.

7.5.2. Conservación de las dependencias

Hay otro objetivo en el diseño de las bases de datos relacionales: la *conservación de las dependencias*. Cuando se lleva a cabo una actualización de la base de datos el sistema debe poder comprobar que la actualización no crea ninguna relación ilegal, es decir, una relación que no satisface todas las dependencias funcionales dadas. Si hay que comprobar de manera eficiente las actualizaciones, se deben diseñar unos esquemas de bases de datos relacionales que permitan la validación de las actualizaciones sin que haga falta calcular las reuniones.

Para decidir si hay que calcular las reuniones para comprobar una actualización hace falta determinar las dependencias funcionales que hay que comprobar verificando cada relación una a una. Sea F un conjunto de dependencias funcionales del esquema R y $R_1, R_2, ..., R_n$ una descomposición de R. La **restricción** de F a R_i es el conjunto F_i de todas las dependencias funcionales de F^+ que *sólo* incluyen atributos de R_i. Dado que todas las dependencias funcionales de una restricción únicamente implican atributos de un esquema de relación, es posible comprobar el cumplimiento de la condición por una dependencia verificando sólo una relación.

Obsérvese que la definición de restricción utiliza todas las dependencias de F^+, no sólo las de F. Por ejem-

plo, supóngase que se tiene $F = \{A \rightarrow B, B \rightarrow C\}$ y que se tiene una descomposición en AC y AB. La restricción de F a AC es, entonces, $A \rightarrow C$, ya que $A \rightarrow C$ se halla en F^+, aunque no se halle en F.

El conjunto de restricciones $F_1, F_2, ..., F_n$ es el conjunto de dependencias que pueden comprobarse de manera eficiente. Ahora cabe preguntarse si es suficiente comprobar sólo las restricciones. Sea $F' = F_1 \cup F_2 \cup ... \cup F_n$. F' es un conjunto de dependencias funcionales del esquema R, pero, en general $F' \neq F$. Sin embargo, aunque $F' \neq F$, puede ocurrir que $F'^+ = F^+$. Si esto último es cierto, entonces cada dependencia de F está lógicamente implicada por F' y, si se comprueba que se satisface F', se habrá comprobado que se satisface F. Se dice que las descomposiciones que tienen propiedad $F'^+ = F^+$ son **descomposiciones que conservan las dependencias**.

La Figura 7.12 muestra un algoritmo para la comprobación de la conservación de las dependencias. La entrada es un conjunto $E = \{R_1, R_2,..., R_n\}$ de esquemas de relaciones descompuestas y un conjunto F de dependencias funcionales. Este algoritmo resulta costoso, ya que exige el cálculo de F^+; se describirá otro algoritmo que es más eficiente después de haber dado un ejemplo de comprobación de la conservación de las dependencias.

Ahora se puede demostrar que la descomposición de *Esquema-empréstito* conserva las dependencias. En lugar de aplicar el algoritmo de la Figura 7.12, se considera una alternativa más sencilla: se considera cada miembro del conjunto de dependencias funcionales F que se exige que se cumplan en *Esquema-empréstito* y se demuestra que cada una de ellas puede comprobarse, como mínimo, en una relación de la descomposición.

- Se puede comprobar la dependencia funcional: *nombre-sucursal* → *ciudad-sucursal activo* utilizando *Esquema-sucursal* = (*nombre-sucursal, ciudad-sucursal, activo*).
- Se puede comprobar la dependencia funcional: *número-préstamo* → *importe nombre-sucursal* utilizando *Esquema-préstamo* = (*nombre-sucursal, número-préstamo, importe*).

```
calcular F+;
for each esquema Ri de E do
    begin
        Fi : = la restricción de F+ a Ri;
    end
F' := ∅
for each restricción Fi do
    begin
      F' = F' ∪ Fi
    end
calcular F'+;
if (F'+ = F+) then return (true)
           else return (false);
```

FIGURA 7.12. Comprobación de la conservación de las dependencias.

Si puede comprobarse cada miembro de F en una de las relaciones de la descomposición, la descomposición conserva las dependencias. Sin embargo, hay casos en los que, aunque la descomposición conserve las dependencias, hay alguna dependencia de F que no puede comprobarse en ninguna relación de la descomposición. Se puede utilizar, por tanto, la prueba alternativa como condición suficiente que resulta sencilla de comprobar; si falla, no se puede concluir que la descomposición no conserve las dependencia; en lugar de eso, habrá que aplicar la prueba general.

Ahora se dará una prueba más eficiente para la conservación de las dependencias, que evita el cálculo de F^+. La idea es comprobar cada dependencia funcional $\alpha \rightarrow \beta$ de F empleando una forma modificada del cierre de los atributos para ver si la descomposición la conserva. Se aplica el siguiente procedimiento a cada $\alpha \rightarrow \beta$ de F.

resultado = α
while (cambios en *resultado*) **do**
  for each R_i de la descomposición
    t = (*resultado* $\cap R_i)^+ \cap R_i$
    resultado = *resultado* $\cup\ t$

El cierre de los atributos está tomado con respecto a las dependencias funcionales de F. Si *resultado* contiene todos los atributos de β, se conserva la dependencia funcional $\alpha \rightarrow \beta$. La descomposición conserva las dependencias si y sólo si se conservan todas las dependencias de F.

Obsérvese que, en lugar de calcular previamente la restricción de F a R_i y utilizarla para el cálculo del cierre de los atributos de *resultado*, se usa el cierre de los atributos en (*resultado* $\cap R_i$) con respecto a F y luego se intersecta con R_i para obtener un resultado equivalente. Este procedimiento tarda un tiempo polinómico, en lugar del tiempo exponencial necesario para calcular F^+.

7.5.3. Repetición de la información

La descomposición de *Esquema-empréstito* no sufre el problema de repetición de la información que se estudió en el Apartado 7.2. En *Esquema-empréstito* era necesario repetir la ciudad y el activo de la sucursal para cada préstamo. La descomposición separa los datos de la sucursal y los del préstamo en relaciones diferentes, con lo que elimina esta redundancia. De manera parecida, se observa que, en *Esquema-empréstito*, si se concede un solo préstamo a varios clientes, hay que repetir el importe del préstamo una vez por cada cliente (así como la ciudad y activo de la sucursal). En la descomposición, la relación del esquema *Esquema-prestatario* contiene la relación *número-préstamo, nombre-cliente*, y no la contiene ningún otro esquema. Por tanto, sólo

se tiene una tupla por préstamo para cada cliente en la relación de *Esquema-prestatario*. En las otras relaciones que implican a *número-préstamo* (las de los esquemas *Esquema-préstamo* y *Esquema-prestatario*) solamente hace falta que aparezca una tupla por préstamo.

Evidentemente, la falta de redundancia de la descomposición es algo deseable. El grado hasta el que se puede conseguir esta falta de redundancia viene representado por varias *formas normales*, que se estudiarán en el resto del capítulo.

7.6. FORMA NORMAL DE BOYCE–CODD

Mediante las dependencias funcionales se pueden definir varias *formas normales* que representan «buenos» diseños de bases de datos. En este apartado se tratará de la FNBC (forma normal de Boyce-Codd, que se define a continuación) y, más adelante, en el Apartado 7.7, se tratará de la 3FN (tercera forma normal).

7.6.1. Definición

Una de las formas normales mas deseables que se pueden obtener es la **forma normal de Boyce-Codd** (FNBC). Un esquema de relación R está en FNBC respecto a un conjunto de dependencias funcionales F si, para todas las dependencias funcionales de F^+ de la forma $\alpha \rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple al menos una de las siguientes condiciones:

- $\alpha \rightarrow \beta$ es una dependencia funcional trivial (es decir, $\beta \subseteq \alpha$)
- α es una superclave del esquema R.

Un diseño de base de datos está en FNBC si cada miembro del conjunto de esquemas de relación que constituye el diseño está en FNBC.

A modo de ejemplo, considérense los siguientes esquemas de relación y sus respectivas dependencias funcionales:

- *Esquema-cliente* = (*nombre-cliente, calle-cliente, ciudad-cliente*)
  nombre-cliente → *calle-cliente ciudad-cliente*
- *Esquema-sucursal* = (*nombre-sucursal, activo, ciudad-sucursal*)
  nombre-sucursal → *activo ciudad-sucursal*
- *Esquema-info-préstamo* = (*nombre-sucursal, nombre-cliente, número-préstamo, importe*)
  número-préstamo → *importe nombre-sucursal*

Puede afirmarse que *Esquema-cliente* está en FNBC. Obsérvese que una clave candidata para el esquema es *nombre-cliente*. Las únicas dependencias funcionales no triviales que se cumplen en *Esquema-cliente* tienen a *nombre-cliente* a la izquierda de la flecha. Dado que *nombre-cliente* es una clave candidata, las dependencias funcionales con *nombre-cliente* en la parte izquierda no violan la definición de FNBC. De manera parecida, se puede demostrar fácilmente que el esquema de relación *Esquema-sucursal* está en FNBC.

El esquema *Esquema-info-préstamo*, sin embargo, *no* está en FNBC. En primer lugar, obsérvese que *número-préstamo* no es una superclave de *Esquema-info-préstamo*, ya que *puede* que haya un par de tuplas que representen a un solo préstamo concedido a dos personas, por ejemplo,

(Centro, Sr. Pinilla, P-44, 1.000)
(Centro, Sra. Pinilla, P-44, 1.000)

Como no se ha relacionado ninguna dependencia funcional que descarte el caso anterior, *número-préstamo* no es una clave candidata. Sin embargo, la dependencia funcional *número-préstamo* → *importe* es de tipo no trivial. Por lo tanto, *Esquema-info-préstamo* no satisface la definición de FNBC.

Se puede afirmar que *Esquema-info-préstamo* no está en una forma normal adecuada, ya que sufre del problema de *repetición de información* que se describió en el Apartado 7.2. Se observa que, si hay varios nombres de clientes asociados a un préstamo, en una relación de *Esquema-info-préstamo* es obligatorio repetir el nombre de la sucursal y el importe una vez por cada cliente. Se puede eliminar esta redundancia rediseñando la base de datos de forma que todos los esquemas estén en FNBC. Una manera de abordar este problema es tomar el diseño que no está en FNBC ya existente como punto de partida y descomponer los esquemas que no estén en FNBC. Considérese la descomposición de *Esquema-info-préstamo* en dos esquemas:

Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
Esquema-prestatario = (*nombre-cliente, número-préstamo*)

Esta descomposición es una descomposición de reunión sin pérdida.

Para determinar si esos esquemas están en FNBC es necesario determinar las dependencias funcionales que se les aplican. En este ejemplo resulta sencillo ver que

número-préstamo → *importe nombre-sucursal*

se aplica a *Esquema-préstamo*, y que sólo se aplican las dependencias funcionales triviales a *Esquema-presta-*

tario. Aunque *número-préstamo* no sea una superclave de *Esquema-info-préstamo*, es una clave candidata para *Esquema-préstamo*. Por tanto, los dos esquemas de la descomposición están en FNBC.

Ahora resulta posible evitar la redundancia en el caso en que haya varios clientes asociados a un mismo préstamo. En la relación de *Esquema-préstamo* hay exactamente una tupla para cada préstamo, y una tupla para cada cliente de cada préstamo en la relación de *Esquema-prestatario*. Por tanto, no hay que repetir el nombre de la sucursal y el importe una vez por cada cliente asociado a un préstamo.

A menudo se puede simplificar la comprobación de una relación para ver si satisface FNBC:

- Para comprobar si la dependencia no trivial $\alpha \rightarrow \beta$ provoca una violación de FNBC hay que calcular α^+ (el cierre de los atributos de α) y comprobar si incluye todos los atributos de R; es decir, si es una superclave de R.
- Para comprobar si el esquema de relación R se halla en FNBC basta con comprobar únicamente las dependencias del conjunto dado F en búsqueda de violaciones de FNBC, en lugar de comprobar todas las dependencias de F^+.

Se puede probar que si ninguna de las dependencias de F provoca una violación de FNBC, entonces ninguna de las dependencias de F^+ la provocará tampoco.

Por desgracia, el último procedimiento no funciona cuando una relación está descompuesta. Es decir, *no* basta con utilizar F al comprobar la relación R_i en la descomposición de R para buscar violaciones de FNBC. Por ejemplo, considérese el esquema de relación R (A,B,C,D,E), con las dependencias funcionales F que contienen $A \rightarrow B$ y $BC \rightarrow D$. Supóngase que estuviera descompuesto en $R_1(A,B)$ y $R_2(A,C,D,E)$. Ahora bien, ninguna de las dependencias de F contiene únicamente atributos de (A,C,D,E), por lo que puede inducir a creer que R_2 satisface FNBC. En realidad, hay una dependencia $AC \rightarrow D$ de F^+ (que puede inferirse empleando la regla de la pseudotransitividad a partir de las dos dependencias de F) que demuestra que R_2 no está en FNBC. Por tanto, puede que se necesite alguna dependencia que esté en F^+, pero que no está en F, para demostrar que la relación descompuesta no está en FNBC.

La prueba alternativa de FNBC resulta a veces más sencilla que el cálculo de todas las dependencias de F^+. Para comprobar si la relación R_i de una descomposición de R está en FNBC hay que aplicar esta prueba:

- En cada subconjunto de atributos a de R_i hay que comprobar que α^+ (el cierre de los atributos de α bajo F) no incluye ningún atributo de $R_i - \alpha$ o que incluye todos los atributos de R_i.

Si algún conjunto de atributos a de R_i viola la condición, considérese la siguiente dependencia funcional, que se puede demostrar que se encuentra en F^+:

$$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i.$$

La dependencia anterior demuestra que R_i viola FNBC y es un «testigo» de esa violación. El algoritmo de descomposición de la FNBC, que se verá en el Apartado 7.6.2, hace uso de este testigo.

7.6.2. Algoritmo de descomposición

Ahora se puede exponer un método general para descomponer los esquemas de relación de manera que satisfagan FNBC. La Figura 7.13 muestra un algoritmo para esta tarea. Si R no está en FNBC se puede descomponer en un conjunto de esquemas en FNBC, $R_1, R_2, ..., R_n$ utilizando este algoritmo. El algoritmo utiliza las dependencias («testigos») que demuestran la violación de FNBC para llevar a cabo la descomposición.

La descomposición que genera este algoritmo no sólo está en FNBC, sino que también es una descomposición de reunión sin pérdida. Para ver el motivo de que el algoritmo genere sólo descomposiciones de reunión sin pérdida hay que observar que, cuando se reemplaza el esquema R_i por $(R_i - \beta)$ y (α, β), se cumple $\alpha \rightarrow \beta$ y $(R_i - \beta) \cap (\alpha, \beta) = \alpha$.

Se aplicará el algoritmo de descomposición FNBC al esquema *Esquema-empréstito* que se empleó en el Apartado 7.2 como ejemplo de mal diseño de base de datos:

Esquema-empréstito = (*nombre-sucursal, ciudad-sucursal, activo, nombre-cliente, número-préstamo, importe*)

El conjunto de dependencias funcionales que se exige que se cumplan en *Esquema-empréstito* es

nombre-sucursal → *activo ciudad-sucursal*
número-préstamo → *importe nombre-sucursal*

Una clave candidata para este esquema es {*número-préstamo, nombre-cliente*}.

Se puede aplicar el algoritmo de la Figura 7.13 al ejemplo *Esquema-empréstito* de la manera siguiente:

```
resultado := {R};
hecho := falso;
calcular F+;
while (not hecho) do
    if (hay un esquema Ri de resultado que no esté en FNBC)
        then begin
                sea α → β una dependencia funcional no trivial
                que se cumple en Ri tal que α → Ri no esté en
                F+ y α ∩ β = ∅;
                resultado := (resultado – Ri) ∪ (Ri – β) ∪ (α, β);
            end
        else hecho := cierto;
```

FIGURA 7.13. Algoritmo de descomposición de FNBC.

- La dependencia funcional

  nombre-sucursal → *activo ciudad-sucursal*

  se cumple en *Esquema-empréstito*, pero *nombre-sucursal* no es una superclave. Por tanto, *Esquema-empréstito* no está en FNBC. Se sustituye *Esquema-empréstito* por

  Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activo*)
  Esquema-info-préstamo = (*nombre-sucursal, nombre-cliente, número-préstamo, importe*)

- Las únicas dependencias funcionales no triviales que se cumplen en *Esquema-sucursal* incluyen a *nombre-sucursal* a la izquierda de la flecha. Como *nombre-sucursal* es una clave de *Esquema-sucursal*, la relación *Esquema-sucursal* está en FNBC.

- La dependencia funcional

  número-préstamo → *importe nombre-sucursal*

  se cumple en *Esquema-info-préstamo*, pero *número-préstamo* no es una clave de *Esquema-info-préstamo*. Se sustituye *Esquema-info-préstamo* por

  Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
  Esquema-prestatario = (*nombre-cliente, número-préstamo*)

- *Esquema-préstamo* y *Esquema-prestatario* están en FNBC.

Por tanto, la descomposición de *Esquema-empréstito* da lugar a tres esquemas de relación *Esquema-sucursal, Esquema-préstamo* y *Esquema-prestatario*, cada uno de los cuales está en FNBC. Estos esquemas de relación son los del Apartado 7.5, donde se demostró que la descomposición resultante es, a un tiempo, una descomposición de reunión sin pérdida y una descomposición que preserva las dependencias.

El algoritmo de descomposición FNBC tarda un tiempo exponencial en el tamaño del esquema inicial, ya que el algoritmo para la comprobación de si la relación de la descomposición satisface FNBC puede tardar un tiempo exponencial. Las notas bibliográficas proporcionan referencias de un algoritmo que puede calcular la descomposición FNBC en un tiempo polinómico. No obstante, el algoritmo puede «sobrenormalizar», es decir, descomponer una relación de manera innecesaria.

7.6.3. Conservación de las dependencias

No todas las descomposiciones FNBC conservan las dependencias. A modo de ejemplo, considérese el esquema de relación

Esquema-asesor = (*nombre-sucursal, nombre-cliente, nombre-asesor*)

que indica que el cliente tiene un «asesor personal» en una sucursal determinada. El conjunto F de dependencias funcionales que se exige que se cumpla en *Esquema-asesor* es

nombre-asesor → *nombre-sucursal*
nombre-sucursal nombre-cliente → *nombre-asesor*

Evidentemente, *Esquema-asesor* no está en FNBC, ya que *nombre-asesor* no es una superclave.

Si se aplica el algoritmo de la Figura 7.13 se obtiene la siguiente descomposición FNBC:

Esquema -asesor-sucursal = (*nombre-asesor, nombre-sucursal*)
Esquema -cliente-asesor = (*nombre-cliente, nombre-asesor*)

Los esquemas descompuestos sólo conservan *nombre-asesor* → *nombre-sucursal* (y las dependencias triviales) pero el cierre de {*nombre-asesor* → *nombre-sucursal*} no incluye *nombre-cliente nombre-sucursal* → *nombre-asesor*. La violación de esta dependencia no puede detectarse a menos que se calcule la reunión.

Para ver el motivo de que la descomposición de *Esquema-asesor* en los esquemas *Esquema—asesor-sucursal* y *Esquema-cliente-asesor* no conserva las dependencias se aplica el algoritmo de la Figura 7.12. Se descubre que las restricciones F_1 y F_2 de F para cada esquema son las siguientes:

F_1 = { *nombre-asesor* → *nombre-sucursal* }
$F_2 = \emptyset$ (en *Esquema-asesor-cliente* solamente se cumplen las dependencias triviales)

(En aras de la brevedad no se muestran las dependencias funcionales triviales.) Resulta evidente que la dependencia *nombre-cliente nombre-sucursal* → *nombre-asesor* no está en $(F_1 \cup F_2)^+$ aunque *sí* está en F^+. Por tanto, $(F_1 \cup F_2)^+ \neq F^+$ y la descomposición no conserva las dependencias.

Este ejemplo demuestra que no todas las descomposiciones FNBC conservan las dependencias. Lo que es más, resulta evidente que *ninguna* descomposición FNBC de *Esquema-asesor* puede conservar *nombre-cliente nombre-sucursal* → *nombre-asesor*. Por tanto, el ejemplo demuestra que no se pueden cumplir siempre los tres objetivos del diseño:

1. Reunión sin pérdida
2. FNBC
3. Conservación de las dependencias

Recuérdese que la reunión sin pérdida es una condición esencial para la descomposición, para evitar la pérdida de información. Por tanto, es obligatorio abandonar la FNBC o la conservación de la dependencia. En el Apartado 7.7 se presenta una forma normal alterna-

tiva, denominada **tercera forma normal**, que es una pequeña relajación de la FNBC; el motivo del empleo de la tercera forma normal es que en ella siempre hay una descomposición que conserva las dependencias.

Hay situaciones en las que hay más de un modo de descomponer un esquema en su FNBC. Puede que algunas de estas descomposiciones conserven las dependencias, mientras que puede que otras no lo hagan. Por ejemplo, supóngase que se tiene un esquema de relación $R(A, B, C)$ con las dependencias funcionales $A \rightarrow B$ y $B \rightarrow C$. A partir de este conjunto se puede obtener la dependencia adicional $A \rightarrow C$. Si se utilizara la dependencia $A \rightarrow B$ (o, de manera equivalente, $A \rightarrow C$) para descomponer R, se acabaría con dos relaciones, $R_1(A, B)$ y $R_2(A, C)$; la dependencia $B \rightarrow C$ no se conservaría.

Si en lugar de eso se empleara la dependencia $B \rightarrow C$ para descomponer R, se acabaría con dos relaciones, $R_1(A, B)$ y $R_2(B, C)$, que están en FNBC, y la descomposición, además, conserva las dependencias. Evidentemente, resulta preferible la descomposición en $R_1(A, B)$ y $R_2(B, C)$. En general, por tanto, el diseñador de la base de datos debería examinar las descomposiciones alternativas y escoger una descomposición que conserve las dependencias siempre que resulte posible.

7.7. TERCERA FORMA NORMAL

Como ya se ha visto, hay esquemas relacionales en que la descomposición FNBC no puede conservar las dependencias. Para estos esquemas hay dos alternativas si se desea comprobar si una actualización viola alguna dependencia funcional:

- Soportar el coste extra del cálculo de las reuniones para buscar violaciones.
- Emplear una descomposición alternativa, la tercera forma normal (3FN), que se presenta a continuación, que hace menos costoso el examen de las actualizaciones. A diferencia de FNBC, las descomposiciones 3FN pueden contener cierta redundancia en el esquema descompuesto.

Se verá que siempre resulta posible hallar una descomposición de reunión sin pérdida que conserve las dependencias que esté en 3FN. La alternativa que se escoja es una decisión de diseño que debe adoptar el diseñador de la base de datos con base en los requisitos de la aplicación.

7.7.1. Definición

FNBC exige que todas las dependencias no triviales sean de la forma $\alpha \rightarrow \beta$ donde α es una superclave. 3FN relaja ligeramente esta restricción permitiendo dependencias funcionales no triviales cuya parte izquierda no sea una superclave.

Un esquema de relación R está en **tercera forma normal (3FN)** respecto a un conjunto F de dependencias funcionales si, para todas las dependencias funcionales de F^+ de la forma $\alpha \rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple al menos una de las siguientes condiciones:

- $\alpha \rightarrow \beta$ es una dependencia funcional trivial.
- α es una superclave de R.
- Cada atributo A de $\beta - \alpha$ está contenido en alguna clave candidata de R.

Obsérvese que la tercera condición no dice que una sola clave candidata deba contener todos los atributos de $\alpha \rightarrow \beta$; cada atributo A de $\alpha \rightarrow \beta$ puede estar contenido en una clave candidata *diferente*.

Las dos primeras alternativas son iguales que las dos alternativas de la definición de FNBC. La tercera alternativa de la definición de 3FN parece bastante intuitiva, y no resulta evidente el motivo de su utilidad. Representa, en cierto sentido, una relajación mínima de las condiciones de FNBC que ayudan a asegurar que cada esquema tenga una descomposición que conserve las dependencias en 3FN. Su finalidad se aclarará más adelante, cuando se estudie la descomposición en 3FN.

Obsérvese que cualquier esquema que satisfaga FNBC satisface también 3FN, ya que cada una de sus dependencias funcionales satisfará una de las dos primeras alternativas. Por tanto, FNBC es una restricción más estricta que 3FN.

La definición de 3FN permite ciertas dependencias funcionales que no se permitían en FNBC. Una dependencia $\alpha \rightarrow \beta$ que sólo satisfaga la tercera alternativa de la definición de 3FN no se permitiría en FNBC, pero sí se permite en 3FN[1].

Volvamos al ejemplo *Esquema-asesor* (Apartado 7.6). Se demostró que este esquema de relación no tiene una descomposición FNBC de reunión sin pérdida que conserve las dependencias. Resulta, sin embargo, que este esquema está en 3FN. Para comprobarlo, obsérvese que {*nombre-cliente, nombre-sucursal*} es una cla-

[1] Estas dependencias son ejemplos de **dependencias transitivas** (véase el Ejercicio 7.25). La definición original de 3FN venía en términos de las dependencias transitivas. La definición que se ha empleado es equivalente, pero más sencilla de comprender.

ve candidata de *Esquema-asesor*, por lo que el único atributo no contenido en una clave candidata de *Esquema-asesor* es *nombre-asesor*. Las únicas dependencias funcionales no triviales de la forma

$$\alpha \rightarrow \textit{nombre-asesor}$$

incluyen {*nombre-cliente, nombre-sucursal*} como parte de α. Dado que {*nombre-cliente, nombre-sucursal*} es una clave candidata, estas dependencias no violan la definición de 3FN.

Como optimización, al buscar 3FN, se pueden considerar sólo las dependencias funcionales del conjunto dado F, en lugar de las de F^+. Además, se pueden descomponer las dependencias de F de manera que su lado derecho consista sólo en atributos sencillos y utilizar el conjunto resultante en lugar de F.

Dada la dependencia $\alpha \rightarrow \beta$, se puede utilizar la misma técnica basada en el cierre de los atributos que se empleó para FNBC para comprobar si α es una superclave. Si α no es una superclave, hay que comprobar si cada atributo de β está contenido en alguna clave candidata de R; esta comprobación resulta bastante más costosa, ya que implica buscar las claves candidatas. De hecho, se ha demostrado que la comprobación de la 3FN resulta NP-duro; por tanto, resulta bastante improbable que haya algún polinomio con complejidad polinómica en el tiempo para esta tarea.

7.7.2. Algoritmo de descomposición

La Figura 7.14 muestra un algoritmo para la búsqueda de descomposiciones de reunión sin pérdida que conserven las dependencias en 3FN. El conjunto de dependencias *Fc* utilizado en el algoritmo es un recubrimiento canónico de F. Obsérvese que el algoritmo considera el conjunto de esquemas R_j, $j = 1, 2, \ldots, i$; inicialmente $i = 0$, y en este caso el conjunto está vacío.

```
sea Fc un recubrimiento canónico de F;
i := 0;
for each dependencia funcional α → β de Fc do
    if ninguno de los esquemas Rj, j = 1,2,..., i contiene αβ
        then begin
                i := i +1;
                Ri := αβ;
            end
if ninguno de los esquemas Rj, j = 1,2,..., i contiene una clave
candidata de R
    then begin
            i := i +1;
            Ri := cualquier clave candidata de R;
        end
return (R1, R2, ..., Ri)
```

FIGURA 7.14. Descomposición de reunión sin pérdida que conserva las dependencias en 3FN.

Para ilustrar el algoritmo de la Figura 7.14 considérese la siguiente extensión de *Esquema-asesor* del Apartado 7.6:

Esquema-info-asesor = (*nombre-sucursal, nombre-cliente, nombre-asesor, número-sucursal*)

La diferencia principal es que se incluye el número de la sucursal del asesor como parte de la información. Las dependencias funcionales para este esquema de relación son:

nombre-asesor → *nombre-sucursal número-sucursal*
nombre-cliente nombre-sucursal → *nombre-asesor*

El bucle **for** del algoritmo hace que se incluyan los siguientes esquemas en la descomposición:

Esquema-asesor-sucursal = (*nombre-asesor, nombre-sucursal, número-sucursal*)
Esquema-asesor = (*nombre-cliente, nombre-sucursal, nombre-asesor*)

Como *Esquema-asesor* contiene una clave candidata de *Esquema-info-asesor*, el proceso de descomposición ha terminado.

El algoritmo asegura la conservación de las dependencias creando de manera explícita un esquema para cada dependencia del recubrimiento canónico. Asegura que la descomposición sea una descomposición de reunión sin pérdida garantizado que, como mínimo, un esquema contenga una clave candidata del esquema que está descomponiendo. El Ejercicio 7.19 proporciona algunos indicios de la prueba de que esto basta para garantizar una reunión sin pérdida.

Este algoritmo también se denomina **algoritmo de síntesis de 3FN**, ya que toma un conjunto de dependencias y añade los esquemas uno a uno, en lugar de descomponer el esquema inicial de manera repetida. El resultado no queda definido de manera única, ya que cada conjunto de dependencias funcionales puede tener más de un recubrimiento canónico y, además, en algunos casos el resultado del algoritmo depende del orden en que considere las dependencias de F_c.

Si una relación R_i está en la descomposición generada por el algoritmo de síntesis, entonces R_i está en 3FN. Recuérdese que, cuando se busca 3FN, basta con considerar las dependencias funcionales cuyo lado derecho sea un solo atributo. Por tanto, para ver si R_i está en 3FN, hay que convencerse de que cualquier dependencia funcional $\gamma \rightarrow B$ que se cumpla en R_i satisface la definición de 3FN. Supóngase que la dependencia que generó R_i en el algoritmo de síntesis es $\alpha \rightarrow \beta$. Ahora bien, B debe estar en α o en β, ya que B está en R_i y $\alpha \rightarrow \beta$ generó R_i. Considérense los tres casos posibles:

- B está tanto en α como en β. En este caso, la dependencia $\alpha \rightarrow \beta$ no habría estado en F_c, ya que B sería raro en β. Por tanto, este caso no puede darse.

- B está en β pero no en α. Considérense dos casos:
  - γ es una superclave. Se satisface la segunda condición de 3FN.
  - γ no es superclave. Entonces α debe contener algún atributo que no se halle en γ. Ahora bien, como $\gamma \rightarrow B$ se halla en F^+, debe poder obtenerse a partir de F_c mediante el algoritmo del cierre de atributos de γ. La obtención no podría haber empleado $\alpha \rightarrow \beta$ –para hacerlo, α debe estar contenido en el cierre de los atributos de γ, lo que no resulta posible, ya que se ha dado por supuesto que γ no es una superclave. Ahora bien, empleando $\alpha \rightarrow (\beta - \{B\})$ y $\gamma \rightarrow B$, se puede obtener $\alpha \rightarrow B$ (ya que $\gamma \subseteq \alpha\beta$ y que γ no puede contener a B porque $\gamma \rightarrow B$ es no trivial). Esto implicaría que B es raro en el lado derecho de $\alpha \rightarrow \beta$, lo que no resulta posible, ya que $\alpha \rightarrow \beta$ está en el recubrimiento canónico F_c. Por tanto, si B está en β, entonces γ debe ser una superclave, y se satisface la segunda condición de 3FN.
- B está en α pero no en β.

  Como $\alpha\beta$ es una clave candidata, se satisface la tercera alternativa de la definición de 3FN.

Resulta de interés que el algoritmo que se ha descrito para la descomposición en 3FN pueda implementarse en tiempo polinómico, aunque la comprobación de una relación dada para ver si satisface 3FN sea NP-duro.

7.7.3. Comparación entre FNBC y 3FN

De las dos formas normales para los esquemas de las bases de datos relacionales, 3FN y FNBC, hay ventajas en 3FN porque se sabe que siempre resulta posible obtener un diseño en 3FN sin sacrificar la reunión sin pérdida o la conservación de las dependencias. Sin embargo, hay inconvenientes en 3FN: si no se eliminan todas las dependencias transitivas de las relaciones de los esquemas, puede que se tengan que emplear valores nulos para representar algunas de las relaciones significativas posibles entre los datos, y está el problema de repetición de la información.

Como ilustración del problema de los valores nulos, considérese de nuevo *Esquema-asesor* y las dependencias funcionales asociadas. Dado que *nombre-asesor* → *nombre-sucursal*, puede que se desee representar las relaciones entre los valores de *nombre-asesor* y los valores de *nombre-sucursal* de la base de datos. No obstante, si se va a hacer eso, o bien debe existir el valor correspondiente de *nombre-cliente* o bien hay que utilizar un valor nulo para el atributo *nombre-cliente*.

Como ilustración del problema de repetición de información, considérese el ejemplo de *Esquema-asesor* de la Figura 7.15. Obsérvese que la información que indica que González trabaja en la sucursal Navacerrada está repetida.

nombre-cliente	*nombre-asesor*	*nombre-sucursal*
Santos	González	Navacerrada
Gómez	González	Navacerrada
López	González	Navacerrada
Sotoca	González	Navacerrada
Pérez	González	Navacerrada
Abril	González	Navacerrada

FIGURA 7.15. Ejemplo de *Esquema-asesor*.

Recuérdese que los objetivos del diseño de bases de datos con dependencias funcionales son:

1. FNBC
2. Reunión sin pérdida
3. Conservación de las dependencias

Como no siempre resulta posible satisfacer las tres, puede que haya que escoger entre FNBC y la conservación de las dependencias con 3FN.

Merece la pena destacar que SQL no proporciona una manera de especificar las dependencias funcionales, salvo para el caso especial de la declaración de las superclaves mediante las restricciones **primary key** o **unique**. Resulta posible, aunque un poco complicado, escribir afirmaciones que hagan que se cumpla una dependencia funcional (véase el Ejercicio 7.15); por desgracia, la comprobación de estas afirmaciones resultaría muy costosa en la mayor parte de los sistemas de bases de datos. Por tanto, aunque se tenga una descomposición que conserve las dependencias, si se utiliza SQL estándar, no se podrá comprobar de manera eficiente las dependencias funcionales cuyo lado izquierdo no sea una clave.

Aunque puede que la comprobación de las dependencias funcionales implique una reunión si la descomposición no conserva las dependencias, se puede reducir el coste empleando las vistas materializadas, que la mayor parte de los sistemas de bases de datos soporta. Dada una descomposición FNBC que no conserve las dependencias, se considera cada dependencia de un recubrimiento mínimo F_c que no se conserva en la descomposición. Para cada una de estas dependencias $\alpha \rightarrow \beta$, se define una vista materializada que calcula una reunión de todas las relaciones de la descomposición y proyecta el resultado sobre $\alpha\beta$. La dependencia funcional puede comprobarse fácilmente en la vista materializada mediante una restricción **única** (α). En el lado negativo hay que contar una sobrecarga espacial y temporal debida a la vista materializada pero, en el lado positivo, el programador de la aplicación no tiene que preocuparse por la escritura de código para hacer que los datos redundantes se conserven consistentes en las actualizaciones; es labor del sistema de bases de datos conservar la vista materializada, es decir, mantenerla actualizada cuando se actualice la base de datos. (Más avanzado el libro, en el Apartado 14.5, se describe el modo en que el sis-

tema de bases de datos puede llevar a cabo de manera eficiente el mantenimiento de las vistas materializadas).

Por tanto, en caso de que no se pueda obtener una descomposición FNBC que conserve las dependencias, suele resultar preferible optar por FNBC y utilizar técnicas como las vistas materializadas para reducir el coste de la comprobación de las dependencias funcionales.

7.8. CUARTA FORMA NORMAL

No parece que algunos esquemas de relación, aunque se hallen en FNBC, estén lo bastante normalizados, en el sentido de que siguen sufriendo el problema de la repetición de la información.

Considérese nuevamente el ejemplo bancario. Supóngase que, en un diseño alternativo del esquema de la base de datos, se tiene el esquema

Esquema-BC = (*número-préstamo, nombre-cliente, calle-cliente, ciudad-cliente*)

El lector sagaz reconocerá este esquema como un esquema que no está en FNBC debido a la dependencia funcional

nombre-cliente → *calle-cliente ciudad-cliente*

que se estableció anteriormente, y debido a que *nombre-cliente* no es una clave de *Esquema-BC*. Sin embargo, supóngase que el banco está atrayendo a clientes ricos que tienen varios domicilios (por ejemplo, una residencia de invierno y otra de verano). Entonces ya no se deseará hacer que se cumpla la dependencia funcional *nombre-cliente* → *calle-cliente ciudad-cliente*. Si se elimina esta dependencia funcional, se halla que *Esquema-BC* está en FNBC con respecto al conjunto modificado de dependencias funcionales. Sin embargo, aunque *Esquema-BC* esté ahora en FNBC, sigue existiendo el problema de la repetición de la información que se tenía anteriormente.

Para tratar este problema hay que definir una nueva forma de restricción, denominada *dependencia multivalorada*. Como se hizo para las dependencias funcionales, se utilizarán las dependencias multivaloradas para definir una forma normal para los esquemas de relación. Esta forma normal, denominada **cuarta forma normal** (4FN), es más restrictiva que FNBC. Se verá que cada esquema 4FN se halla también en FNBC, pero que hay esquemas FNBC que no se hallan en 4FN.

7.8.1. Dependencias multivaloradas

Las dependencias funcionales impiden que ciertas tuplas estén en una relación. Si $A \rightarrow B$, entonces no puede haber dos tuplas con el mismo valor de A y diferentes valores de B. Las dependencias multivaloradas, por otro lado, no impiden la existencia de esas tuplas. En lugar de eso, *exigen* que estén presentes en la relación otras tuplas de una cierta forma. Por este motivo, las dependencias funcionales se denominan a veces **dependencias de generación de igualdad** y las dependencias multivaloradas se conocen como **dependencias de generación de tuplas**.

Sea R un esquema de relación y sean $\alpha \subseteq R$ y $\beta \subseteq R$. La **dependencia multivalorada**

$$\alpha \twoheadrightarrow \beta$$

se cumple en R si, en toda relación legal $r(R)$, para todo par de tuplas t_1 y t_2 de r tales que $t_1[\alpha] = t_2[\alpha]$, existen unas tuplas t_3 y t_4 de r tales que

$$\begin{aligned}
&t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]\\
&t_3[\beta] = t_1[\beta]\\
&t_3[R - \beta] = t_2[R - \beta]\\
&t_4[\beta] = t_2[\beta]\\
&t_4[R - \beta] = t_1[R - \beta]
\end{aligned}$$

Esta definición es menos complicada de lo que parece. La Figura 7.16 muestra una representación tabular de t_1, t_2, t_3 y t_4. De manera intuitiva, la dependencia multivalorada $\alpha \twoheadrightarrow \beta$ indica que la relación entre α y β es independiente de la relación entre α y $R - \beta$. Si todas las relaciones del esquema R satisfacen la dependencia multivalorada $\alpha \twoheadrightarrow \beta$, entonces $\alpha \twoheadrightarrow \beta$ es una dependencia multivalorada *trivial* en el esquema R. Por tanto, $\alpha \twoheadrightarrow \beta$ es trivial si $\beta \subseteq \alpha$ o $\beta \cup \alpha = R$.

Para ilustrar la diferencia entre las dependencias funcionales y las multivaloradas, considérense de nuevo *Esquema-BC* y la relación *bc* (*Esquema-BC*) de la Figura 7.17. Hay que repetir el número de préstamo una vez por cada dirección que tenga un cliente, y la dirección en cada préstamo que tenga el cliente. Esta repetición es innecesaria, ya que la relación entre el cliente y su dirección es independiente de la relación entre ese cliente y el préstamo. Si un cliente (por ejemplo, Gómez) tiene un préstamo (por ejemplo, el préstamo número P-23), se desea que ese préstamo esté asociado con todas las direcciones que tenga Gómez. Por tanto, la relación

	α	β	$R - \alpha - \beta$
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$

FIGURA 7.16. Representación tabular de $\alpha \twoheadrightarrow \beta$.

número-préstamo	nombre-cliente	calle-cliente	ciudad-cliente
P-23	Gómez	Carretas	Cerceda
P-23	Gómez	Mayor	Chinchón
P-93	Pérez	Leganitos	Aluche

FIGURA 7.17. Relación *bc*: ejemplo de redundancia en una relación FNBC.

de la Figura 7.18 es ilegal. Para hacer que esta relación sea legal hay que añadir las tuplas (P-23, Gómez, Mayor, Chinchón) y (P-27, Gómez, Carretas, Cerceda) a la relación *bc* de la Figura 7.18.

Si se compara el ejemplo anterior con la definición de dependencia multivalorada, se ve que se desea que se cumpla la dependencia multivalorada

$$nombre\text{-}cliente \rightarrow\!\!\!\rightarrow calle\text{-}cliente\ ciudad\text{-}cliente$$

(La dependencia multivalorada *nombre-cliente* $\rightarrow\!\!\!\rightarrow$ *número-préstamo* también se cumplirá. Pronto se verá que son equivalentes.)

Al igual que con las dependencias funcionales, las dependencias multivaloradas se utilizan de dos maneras:

1. Para verificar las relaciones y determinar si son legales bajo un conjunto dado de dependencias funcionales y multivaloradas.
2. Para especificar restricciones del conjunto de relaciones legales; de este modo, *sólo* habrá que preocuparse de las relaciones que satisfagan un conjunto dado de dependencias funcionales y multivaloradas.

Obsérvese que, si una relación r no satisface una dependencia multivalorada dada, se puede crear una relación r' que *sí* satisfaga esa dependencia multivalorada añadiendo tuplas a r.

Supongamos que F denota un conjunto de dependencias funcionales y multivaloradas. El **cierre** F^+ de F es el conjunto de todas las dependencias funcionales y multivaloradas implicadas lógicamente por F. Al igual que se hizo para las dependencias funcionales, se puede calcular F^+ a partir de F, empleando las definiciones formales de las dependencias funcionales y de las dependencias multivaloradas. Con este razonamiento se puede trabajar con las dependencias multivaloradas muy sencillas. Afortunadamente, parece que las dependencias multivaloradas que se dan en la práctica son bastante sencillas. Para las dependencias complejas es mejor razonar con conjuntos de dependencias empleando un sistema de reglas de inferencia. (El Apartado C.1.1 del apéndice describe un sistema de reglas de inferencia para las dependencias multivaloradas.)

número-préstamo	nombre-cliente	calle-cliente	ciudad-cliente
P-23	Gómez	Carretas	Cerceda
P-27	Gómez	Mayor	Chinchón

FIGURA 7.18. Una relación *bc* ilegal.

A partir de la definición de dependencia multivalorada se puede obtener la regla siguiente:

- Si $\alpha \rightarrow \beta$, entonces $\alpha \rightarrow\!\!\!\rightarrow \beta$.

En otras palabras, cada dependencia funcional es también una dependencia multivalorada.

7.8.2. Definición de la cuarta forma normal

Considérese nuevamente el ejemplo del *Esquema-BC* en el que se cumple la dependencia multivalorada *nombre-cliente* $\rightarrow\!\!\!\rightarrow$ *calle-cliente ciudad-cliente*, pero no se cumple ninguna dependencia funcional no trivial. Se vio en los primeros párrafos del Apartado 7.8 que, aunque *Esquema-BC* se halla en FNBC, el diseño no es el ideal, ya que hay que repetir la información de la dirección del cliente para cada préstamo. Se verá que se puede utilizar la dependencia multivalorada dada para mejorar el diseño de la base de datos descomponiendo *Esquema-BC* en una descomposición en la **cuarta forma normal**.

Un esquema de relación R está en la **cuarta forma normal** (4FN) con respecto a un conjunto F de dependencias funcionales y multivaloradas si, para todas las dependencias multivaloradas de F^+ de la forma $\alpha \rightarrow\!\!\!\rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple, como mínimo, una de las condiciones siguientes

- $\alpha \rightarrow\!\!\!\rightarrow \beta$ es una dependencia multivalorada trivial.
- α es una superclave del esquema R.

Un diseño de base de datos está en 4FN si cada componente del conjunto de esquemas de relación que constituye el diseño se halla en 4FN.

Obsérvese que la definición de 4FN sólo se diferencia de la definición de FNBC en el empleo de las dependencias multivaloradas en lugar de las dependencias funcionales. Todos los esquemas 4FN están en FNBC. Para verlo hay que darse cuenta de que, si un esquema R no se halla en FNBC, hay una dependencia funcional no trivial $\alpha \rightarrow \beta$ que se cumple en R, donde α no es una superclave. Como $\alpha \rightarrow \beta$ implica $\alpha \rightarrow\!\!\!\rightarrow \beta$, R no puede estar en 4FN.

Sea R un algoritmo de descomposición y sea R_1, R_2, . . . , R_n una descomposición de R. Para comprobar si cada esquema de relación R_i se halla en 4FN, hay que averiguar las dependencias multivaloradas que se cumplen en cada R_i. Recuérdese que, para un conjunto F de dependencias funcionales, la restricción F_i de F a R_i son todas las dependencias funcionales de F^+ que *sólo* incluyen los atributos de R_i. Considérese ahora un conjunto F de dependencias funcionales y multivaloradas. La **restricción** de F a R_i es el conjunto F_i que consta de

1. Todas las dependencias funcionales de F^+ que sólo incluyen atributos de R_i
2. Todas las dependencias multivaloradas de la forma

$$\alpha \twoheadrightarrow \beta \cap R_i$$

donde $\alpha \subseteq R_i$ y $\alpha \twoheadrightarrow \beta$ está en F^+.

7.8.3. Algoritmo de descomposición

La analogía entre 4FN y FNBC es aplicable al algoritmo para la descomposición de los esquemas en 4FN. La Figura 7.19 muestra el algoritmo de descomposición en 4FN. Es idéntico al algoritmo de descomposición en FNBC de la Figura 7.13, excepto en que emplea dependencias multivaloradas en lugar de funcionales y en que utiliza la restricción de F^+ a R_i.

Si se aplica el algoritmo de la Figura 7.19 a *Esquema-BC* se halla que *nombre-cliente* $\twoheadrightarrow$ *número-préstamo* es una dependencia multivalorada no trivial, y que *nombre-cliente* no es una superclave de *Esquema-BC*. Siguiendo el algoritmo, se sustituye *Esquema-BC* por dos esquemas:

Esquema-prestatario = (*nombre-cliente*, *número-préstamo*)
Esquema-cliente = (*nombre-cliente, calle-cliente, ciudad-cliente*).

Este par de esquemas, que se halla en 4FN, elimina el problema que se encontró anteriormente con la redundancia de *Esquema-BC*.

Como ocurría cuando se trataba solamente con las dependencias funcionales, resultan interesantes las descomposiciones que son descomposiciones de reunión sin pérdida y que conservan las dependencias. El hecho siguiente relativo a las dependencias multivaloradas y las reuniones sin pérdida muestra que el algoritmo de la Figura 7.19 sólo genera descomposiciones de reunión sin pérdida:

```
resultado := {R};
hecho := falso;
calcular F+; Dado el esquema Ri, supongamos que Fi denota la
restricción de F+ a Ri
while (not hecho) do
    if (hay un esquema Ri en resultado que no se halla en 4FN
con respecto a Fi)
        then begin
                supongamos que α ->> β es una dependencia
                multivalorada no trivial que se cumple en Ri tal
                que α -> Ri no se halla en Fi, y α ∩ β = ∅;
                resultado := (resultado – Ri) ∪ (Ri – β) ∪ (α, β);
            end
        else hecho := verdadero;
```

FIGURA 7.19. Alogaritmo de descomposición en 4FN.

- Sean R un esquema de relación y F un conjunto de dependencias funcionales y multivaloradas de R. Supongamos que R_1 y R_2 forman una descomposición de R. Esta descomposición será una descomposición de reunión sin pérdida de R si y sólo si, como mínimo, una de las siguientes dependencias multivaloradas se halla en F^+:

$$R_1 \cap R_2 \twoheadrightarrow R_1$$
$$R_1 \cap R_2 \twoheadrightarrow R_2$$

Recuérdese que se afirmó en el Apartado 7.5.1 que, si $R_1 \cap R_2 \rightarrow R_1$ o $R_1 \cap R_2 \rightarrow R_2$, entonces R_1 y R_2 son una descomposición de reunión sin pérdida de R. El hecho anterior sobre las dependencias multivaloradas es una afirmación más general sobre las reuniones sin pérdida. Dice que, para *toda* descomposición de reunión sin pérdida de R en dos esquemas R_1 y R_2, debe cumplirse una de las dos dependencias $R_1 \cap R_2 \twoheadrightarrow R_1$ o $R_1 \cap R_2 \twoheadrightarrow R_2$.

El problema de la conservación de la dependencia al descomponer una relación se vuelve más complejo en presencia de dependencias multivaloradas. El Apartado C.1.2 del apéndice aborda este tema.

7.9. OTRAS FORMAS NORMALES

La cuarta forma normal no es, de ningún modo, la forma normal «definitiva». Como ya se ha visto, las dependencias multivaloradas ayudan a comprender y a abordar algunas formas de repetición de la información que no pueden comprenderse en términos de las dependencias funcionales. Hay tipos de restricciones denominadas **dependencias de reunión** que generalizan las dependencias multivaloradas y llevan a otra forma normal denominada **forma normal de reunión por proyección (FNRP)** (la FNRP se denomina en algunos libros **quinta forma normal**). Hay una clase de restricciones todavía más generales, que lleva a una forma normal denominada **forma normal de dominios y claves (FNDC)**.

Un problema práctico del empleo de estas restricciones generalizadas es que no sólo es difícil razonar con ellas, sino que tampoco hay un conjunto de reglas de inferencia seguras y completas para razonar sobre las restricciones. Por tanto, la FNRP y la forma normal de dominios y claves se utilizan muy raramente. El Apéndice C ofrece más detalles sobre estas formas normales.

Conspicua por su ausencia de este estudio de las formas normales es la **segunda forma normal** (2FN). No se ha estudiado porque sólo es de interés histórico. Simplemente se definirá, y se permitirá al lector experimentar con ella en el Ejercicio 7.26.

7.10. PROCESO GENERAL DEL DISEÑO DE BASES DE DATOS

Hasta ahora se han examinado problemas concretos de las formas normales y de la normalización. En este apartado se estudiará el modo en que se encaja la normalización en proceso general de diseño de bases de datos.

Anteriormente, en este capítulo, a partir del Apartado 7.4, se dio por supuesto que se da un esquema de relación *R* y que se procede a normalizarlo. Hay varios modos de obtener el esquema *R*:

1. *R* puede haberse generado al convertir un diagrama E-R en un conjunto de tablas.
2. *R* puede haber sido una sola relación que contuviera *todos* los atributos que resultan de interés. El proceso de normalización divide a *R* en relaciones más pequeñas.
3. *R* puede haber sido el resultado de algún diseño ad hoc de relaciones, que hay que comprobar para verificar que satisface la forma normal deseada.

En el resto de este apartado se examinarán las implicaciones de estos enfoques. También se examinarán algunos problemas prácticos del diseño de bases de datos, incluida la desnormalización para el rendimiento y ejemplos de mal diseño que no detecta la normalización.

7.10.1. El modelo E-R y la normalización

Cuando se define con cuidado un diagrama E-R, identificando correctamente todas las entidades, las tablas generadas a partir del diagrama E-R no necesitan más normalización. No obstante, puede haber dependencias funcionales entre los atributos de una entidad. Por ejemplo, supóngase que una entidad *empleado* tiene los atributos *número-departamento* y *dirección-departamento*, y que hay una dependencia funcional *número-departamento* → *dirección-departamento*. Habrá que normalizar la relación generada a partir de *empleado*.

La mayor parte de los ejemplos de estas dependencias surgen de un mal diseño del diagrama E-R. En el ejemplo anterior, si se hiciera correctamente el diagrama E-R, se habría creado una entidad *departamento* con el atributo *dirección-departamento* y una relación entre *empleado* y *departamento*. De manera parecida, puede que una relación que implique a más de dos entidades no se halle en una forma normal deseable. Como la mayor parte de las relaciones son binarias, estos casos resultan relativamente raros. (De hecho, algunas variantes de los diagramas E-R hacen realmente difícil o imposible especificar relaciones no binarias.)

Las dependencias funcionales pueden ayudar a detectar el mal diseño E-R. Si las relaciones generadas no se hallan en la forma normal deseada, el problema puede solucionarse en el diagrama E-R. Es decir, la normalización puede llevarse a cabo formalmente como parte del modelado de los datos. De manera alternativa, la normalización puede dejarse a la intuición del diseñador durante el modelado E-R, y puede hacerse formalmente sobre las relaciones generadas a partir del modelo E-R.

7.10.2. El enfoque de la relación universal

El segundo enfoque del diseño de bases de datos es comenzar con un solo esquema de relación que contenga todos los atributos de interés y descomponerlo. Uno de los objetivos al escoger una descomposición era que fuera una descomposición de reunión sin pérdida. Para considerar la carencia de pérdida se dio por supuesto que resulta válido hablar de la reunión de todas las relaciones de la base de datos descompuesta.

Considérese la base de datos de la Figura 7.20, que muestra una descomposición de la relación *info-préstamo*. La figura muestra una situación en la que no se ha determinado todavía el importe del préstamo P-58, pero se desea registrar el resto de los datos del préstamo. Si se calcula la reunión natural de estas relaciones, se descubre que todas las tuplas que hacen referencia al préstamo P-58 desaparecen. En otras palabras, no hay relación *info-préstamo* correspondiente a las relaciones de la Figura 7.20. Las tuplas que desaparecen cuando se calcula la reunión son *tuplas colgantes* (véase el Apartado 6.2.1). Formalmente, sea $r_1(R_1)$, $r_2(R_2)$, . . . , $r_n(R_n)$ un conjunto de relaciones. Una tupla *t* de la relación r_i es una tupla colgante si *t* no está en la relación

$$\Pi_{R_i}(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n)$$

Las tuplas colgantes pueden aparecer en las aplicaciones prácticas de las bases de datos. Representan información incompleta, como en el ejemplo, en que se desea almacenar datos sobre un préstamo que todavía se halla en proceso de negociación. La relación $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$ se denomina **relación universal**, ya que implica a todos los atributos del universo definido por $R_1 \cup R_2 \cup \cdots \cup R_n$.

nombre-sucursal	*número-préstamo*
Collado Mediano	P-58

número- préstamo	*importe*

número-préstamo	*nombre-cliente*
P-58	González

FIGURA 7.20. Descomposición de *info-préstamo*.

La única manera de escribir una relación universal para el ejemplo de la Figura 7.20 es incluir los *valores nulos* en la relación universal. Se vio en el Capítulo 3 que los valores nulos presentan varias dificultades. Debido a ello, puede que sea mejor ver las relaciones del diseño descompuesto como representación de la base de datos, en lugar de como la relación universal cuyo esquema se descompuso durante el proceso de normalización. (Las notas bibliográficas discuten la investigación sobre los valores nulos y las relaciones universales.)

Téngase en cuenta que no se puede introducir en la base de datos de la Figura 7.20 toda la información incompleta sin recurrir a los valores nulos. Por ejemplo, no se puede introducir un número de préstamo a menos que se conozca, como mínimo, uno de los datos siguientes:

- El nombre del cliente
- El nombre de la sucursal
- El importe del préstamo

Por tanto, cada descomposición concreta define una forma restringida de información incompleta que es aceptable en la base de datos.

Las formas normales que se han definido generan buenos diseños de bases de datos desde el punto de vista de la representación de la información incompleta. Volviendo una vez más al ejemplo de la Figura 7.20, no sería deseable permitir el almacenamiento del hecho siguiente: «Hay un préstamo (cuyo número se desconoce) para Santos con un importe de 100 €». Esto se debe a que

$$\textit{número-préstamo} \rightarrow \textit{nombre-cliente importe}$$

y, por tanto, la única manera de relacionar *nombre-cliente* e *importe* es mediante *número-préstamo*. Si no se conoce el número del préstamo, no se puede distinguir este préstamo de otros préstamos con números desconocidos.

En otras palabras, no se desea almacenar datos de los cuales se desconocen los atributos claves. Obsérvese que las formas normales que se han definido no nos permiten almacenar ese tipo de información a menos que se utilicen los valores nulos. Por tanto, las formas normales permiten la representación de la información incompleta aceptable mediante las tuplas colgantes, mientras que prohíben el almacenamiento de la información incompleta indeseable.

Otra consecuencia del enfoque de la relación universal del diseño de bases de datos es que los nombres de los atributos deben ser únicos en la relación universal. No se puede utilizar *nombre* para hacer referencia tanto a *nombre-cliente* como a *nombre-sucursal*. Generalmente resulta preferible utilizar nombres únicos, como se ha hecho aquí. No obstante, si se definen de manera directa los esquemas de las relaciones, en vez de en términos de una relación universal, se pueden obtener relaciones en esquemas como los siguientes para el ejemplo del banco:

sucursal-préstamo (*nombre, número*)
préstamo-cliente (*número, nombre*)
importe (*número, importe*)

Obsérvese que, con las relaciones anteriores, expresiones como *sucursal-préstamo* ⋈ *préstamo-cliente* carecen de sentido. En realidad, la expresión *sucursal-préstamo* ⋈ *préstamo-cliente* halla los préstamos concedidos por las sucursales a los clientes que tienen el mismo nombre que la sucursal.

En un lenguaje como SQL, sin embargo, una consulta que implique a *sucursal-préstamo* y a *préstamo-cliente* debe eliminar la ambigüedad en las referencias a *nombre* anteponiendo el nombre de la relación. En estos entornos los diferentes papeles de *nombre* (como nombre de la sucursal y del cliente) resultan menos problemáticos y puede que sean más sencillos de utilizar.

Es opinión de los autores de este libro que la **suposición de un único papel** –que cada nombre de atributo tenga un significado único en la base de datos– suele resultar preferible a la reutilización del mismo nombre en varios papeles. Cuando no se realiza la suposición, el diseñador de la base de datos debe ser especialmente cuidadoso al crear un diseño normalizado de una base de datos relacional.

7.10.3. Desnormalización para el rendimiento

A veces los diseñadores de bases de datos escogen un esquema que tiene información redundante; es decir, que no está normalizada. Utilizan la redundancia para mejorar el rendimiento para aplicaciones concretas. La penalización sufrida por no emplear un esquema normalizado es el trabajo extra (en términos de tiempo de codificación y de tiempo de ejecución) de mantener consistentes los datos redundantes.

Por ejemplo, supóngase que hay que mostrar el nombre del titular de una cuenta junto con el número y el saldo de su cuenta cada vez que se tiene acceso a la cuenta. En el esquema normalizado esto exige una reunión de *cuenta* con *impositor*.

Una alternativa para calcular la reunión sobre la marcha es almacenar una relación que contenga todos los atributos de *cuenta* y de *impositor*. Esto hace más rápida la visualización de la información de la cuenta. No obstante, la información del saldo de la cuenta se repite para cada persona titular de la cuenta, y la aplicación debe actualizar todas las copias, siempre que se actualice el saldo de la cuenta. El proceso de tomar un esquema normalizado y hacerlo no normalizado se denomina **desnormalización**, y los diseñadores lo utilizan para ajustar el rendimiento de los sistemas para dar soporte a las operaciones críticas en el tiempo.

Una alternativa mejor, soportada hoy en día por muchos sistemas de bases de datos, es emplear el esquema normalizado y, de manera adicional, almacenar la reunión o *cuenta* e *impositor* en forma de vista materializada. (Recuérdese que una vista materializada es una vista cuyo resultado se almacena en la base de datos y se actualiza cuando se actualizan las relaciones utilizadas en la vista.) Al igual que la desnormalización, el empleo de las vistas materializadas supone sobrecargas de espacio y de tiempo; sin embargo, presenta la ventaja de que conservar la vista actualizada es labor del sistema de bases de datos, no del programador de la aplicación.

7.10.4. Otros problemas de diseño

Hay algunos aspectos del diseño de bases de datos que la normalización no aborda y, por tanto, pueden llevar a un mal diseño de la base de datos. A continuación se ofrecerán algunos ejemplos; evidentemente, conviene evitar esos diseños.

Considérese la base de datos de una empresa, donde se desea almacenar los beneficios de las compañías de varios años. Se puede utilizar la relación *beneficios*(*id-empresa, año, importe*) para almacenar la información de los beneficios. La única dependencia funcional de esta relación es *id-empresa año* → *importe*, y esta relación se halla en FNBC.

Un diseño alternativo es el empleo de varias relaciones, cada una de las cuales almacena los beneficios de un año diferente. Supóngase que los años de interés son 2000, 2001 y 2002; se tendrán, entonces, las relaciones de la forma *beneficios-2000*, *beneficios-2001*, *beneficios-2002*, todos los cuales se hallan en el esquema (*id-empresa, beneficios*). Aquí, la única dependencia funcional de cada relación será *id-empresa* → *beneficios*, por lo que estas relaciones también se hallan en FNBC.

No obstante, este diseño alternativo es, claramente, una mala idea: habría que crear una relación nueva cada año, y también habría que escribir consultas nuevas cada año, para tener en cuenta cada nueva relación. Las consultas también tendrían que ser más complicadas, ya que puede que tengan que hacer referencia a muchas relaciones.

Otra manera más de representar los mismos datos es tener una sola relación *empresa-año*(*id-empresa, beneficios-2000, beneficios-2001, beneficios-2002*). En este caso, las únicas dependencias funcionales van de *id-empresa* hacia los demás atributos, y la relación vuelve a estar en FNBC. Este diseño también es una mala idea, ya que tiene problemas parecidos al diseño anterior, es decir, habría que modificar el esquema de la relación y escribir consultas nuevas cada año. Las consultas también serían más complicadas, ya que puede que tengan que hacer referencia a muchos atributos.

Las representaciones como las de la compañía *empresa-año*, con una columna para cada valor de un atributo, se denominan **de tablas cruzadas**; se emplean ampliamente en las hojas de cálculo, en los informes y en las herramientas de análisis de datos. Aunque estas representaciones resultan útiles para mostrárselas a los usuarios, por las razones que acaban de darse, no resultan deseables en el diseño de bases de datos. Se han propuesto extensiones de SQL para convertir los datos desde una representación relacional normal en una tabla cruzada, para poder mostrarlos.

7.11. RESUMEN

- Se han mostrado algunas dificultades del diseño de bases de datos y el modo de diseñar de manera sistemática esquemas de bases de datos que eviten esas dificultades. Entre esas dificultades están la información repetida y la imposibilidad de representar cierta información.
- Se ha introducido el concepto de las dependencias funcionales y se ha mostrado el modo de razonar con ellas. Se ha puesto un énfasis especial en que las dependencias están implicadas lógicamente por un conjunto de dependencias. También se ha definido el concepto de recubrimiento canónico, que es un conjunto mínimo de dependencias funcionales equivalente a un conjunto dado de dependencias funcionales.
- Se ha introducido el concepto de descomposición y se ha mostrado que las descomposiciones deben ser descomposiciones de reunión sin pérdida y que, preferiblemente, deben conservar las dependencias.
- Si la descomposición conserva las dependencias, dada una actualización de la base de datos, todas las dependencias funcionales pueden verificarse a partir de las relaciones individuales, sin calcular la reunión de las relaciones en la descomposición.
- Luego se ha presentado la Forma normal de Boyce–Codd (FNBC); las relaciones en FNBC están libres de las dificultades ya descritas. Se ha descrito un algoritmo para la descomposición de las relaciones en FNBC. Hay relaciones para las que no hay ninguna descomposición FNBC que conserve las dependencias.
- Se han utilizado los recubrimientos canónicos para descomponer una relación en 3FN, que es una pequeña relajación de la condición FNBC. Puede que las relaciones en 3FN tengan alguna redundancia, pero siempre hay una descomposición en 3FN que conserve las dependencias.

- Se ha presentado el concepto de las dependencias multivaloradas, que especifican las restricciones que no pueden especificarse únicamente con las dependencias funcionales. Se ha definido la cuarta forma normal (4FN) con las dependencias multivaloradas. El Apartado C.1.1 del apéndice da detalles del razonamiento sobre las dependencias multivaloradas.
- Otras formas normales, como FNRP y FNDC, eliminan formas más sutiles de redundancia. No obstante, es difícil trabajar con ellas y se emplean rara vez. El Apéndice C ofrece detalles de estas formas normales.
- Al revisar los temas de este capítulo hay que tener en cuenta que el motivo de que se hayan podido definir enfoques rigurosos del diseño de bases de datos relacionales es que el modelo de datos relacionales descansa sobre una base matemática sólida. Ésa es una de las principales ventajas del modelo relacional en comparación con los otros modelos de datos que se han estudiado.

TÉRMINOS DE REPASO

- Algoritmo de descomposición 3FN
- Algoritmo de descomposición FNBC
- Atributos raros
- Axiomas de Armstrong
- Cierre de un conjunto de dependencias funcionales
- Cierre de los conjuntos de atributos
- Conservación de las dependencias
- Cuarta forma normal
- Dependencias funcionales
- Dependencias funcionales triviales
- Dependencias multivaloradas
- Descomposición
- Descomposición de reunión sin pérdida
- Desnormalización
- Dificultades en el diseño de bases de datos relacionales
- Dominios atómicos
- F se cumple en R
- Forma normal de Boyce–Codd (FNBC)
- Forma normal de reunión por proyección (FNRP)
- Forma normal dominios y claves
- Modelo E-R y normalización
- Primera forma normal
- R satisface F
- Recubrimiento canónico
- Relación universal
- Relaciones legales
- Restricción de F a R_i
- Restricción de las dependencias multivaloradas
- Superclave
- Suposición de un papel único
- Tercera forma normal

EJERCICIOS

7.1. Explíquese lo que se quiere decir con *repetición de la información* e *imposibilidad de representación de la información*. Explíquese el motivo por el que estas propiedades pueden indicar un mal diseño de bases de datos relacionales.

7.2. Supóngase que se descompone el esquema $R = (A, B, C, D, E)$ en

$$(A, B, C)$$
$$(A, D, E)$$

Demuéstrese que esta descomposición es una descomposición de reunión sin pérdida si se cumple el siguiente conjunto F de dependencias funcionales:

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

7.3. Indíquese el motivo de que ciertas dependencias funcionales se denominen dependencias funcionales *triviales*.

7.4. Indíquense todas las dependencias funcionales que satisface la relación de la Figura 7.21.

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

FIGURA 7.21. La relación del Ejercicio 7.4.

7.5. Utilícese la definición de dependencia funcional para argumentar que cada uno de los axiomas de Armstrong (reflexividad, aumentatividad y transitividad) es correcta.

7.6. Explíquese el modo en que las dependencias funcionales pueden utilizarse para indicar lo siguiente:

- Existe un conjunto de relaciones de uno a uno entre los conjuntos de entidades *cuenta* y *cliente*.
- Existe un conjunto de relaciones de varios a uno entre los conjuntos de entidades *cuenta* y *cliente*.

7.7. Considérese la siguiente regla propuesta para las dependencias funcionales: Si $\alpha \rightarrow \beta$ y $\gamma \rightarrow \beta$, entonces $\alpha \rightarrow \gamma$. Pruébese que esta regla *no* es segura mostrando una relación *r* que satisfaga $\alpha \rightarrow \beta$ y $\gamma \rightarrow \beta$, pero no satisfaga $\alpha \rightarrow \gamma$.

7.8. Utilícense los axiomas de Armstrong para probar la seguridad de la regla de la unión. (*Sugerencia*: utilícese la regla de la aumentatividad para probar que, si $\alpha \rightarrow \beta$, entonces $\alpha \rightarrow \alpha\beta$. Aplíquese nuevamente la regla de aumentatividad, utilizando $\alpha \rightarrow \gamma$, y aplíquese luego la regla de transitividad.)

7.9. Utilícense los axiomas de Armstrong para probar la corrección de la regla de la descomposición.

7.10. Utilícense los axiomas de Armstrong para probar la corrección de la regla de la pseudotransitividad.

7.11. Calcúlese el cierre del siguiente conjunto *F* de relaciones funcionales para el esquema de relación $R = (A, B, C, D, E)$.

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

Indíquense las claves candidatas de *R*.

7.12. Utilizando las dependencias funcionales del Ejercicio 7.11, calcúlese B^+.

7.13. Utilizando las dependencias funcionales del Ejercicio 7.11, calcúlese el recubrimiento canónico F_c.

7.14. Considérese el algoritmo de la Figura 7.22 para calcular α^+. Demuéstrese que este algoritmo resulta más eficiente que el presentado en la Figura 7.7 (Apartado 7.3.3) y que calcula α^+ de manera correcta.

7.15. Dado el esquema de base de datos *R* (*a*, *b*, *c*) y una relación *r* del esquema *R*, escríbase una consulta SQL para comprobar si la dependencia funcional $b \rightarrow c$ se cumple en la relación *r*. Escríbase también una declaración SQL que haga que se cumpla la dependencia funcional. Supóngase que no hay ningún valor nulo.

7.16. Demuéstrese que la siguiente descomposición del esquema *R* del Ejercicio 7.2 no es una descomposición de reunión sin pérdida:

$$(A, B, C)$$
$$(C, D, E)$$

Sugerencia: dese un ejemplo de una relación *r* del esquema *R* tal que

$$\Pi_{A,\,B,\,C}(r) \bowtie \Pi_{C,\,D,\,E}(r) \neq r$$

7.17. Sea $R_1, R_2, \ldots, R_n$ una descomposición del esquema *U*. Sea *u*(*U*) una relación y sea $r_i = \Pi_{R_i}(u)$. Demuéstrese que

$$u \subseteq \bowtie\ r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$$

7.18. Demuéstrese que la descomposición del Ejercicio 7.2 no es una descomposición que conserve las dependencias.

7.19. Demuéstrese que es posible que una descomposición que conserve las dependencias en 3FN sea una descomposición de reunión sin pérdida garantizando que, como mínimo, un esquema contenga una clave candidata para el esquema que se está descomponiendo. (*Sugerencia*: demuéstrese que la reunión de todas las proyecciones en los esquemas de la descomposición no puede tener más tuplas que la relación original.)

7.20. Indíquense los tres objetivos de diseño de las bases de datos relacionales y explíquese el motivo de que cada uno de ellos sea deseable.

7.21. Dese una descomposición de reunión sin pérdida en FNBC del esquema *R* del Ejercicio 7.2.

7.22. Dese un ejemplo de esquema de relación R' y de un conjunto F' de dependencias funcionales tales que haya, al menos, tres descomposiciones de reunión sin pérdida distintas de R' en FNBC.

7.23. Al diseñar una base de datos relacional, indíquese el motivo de que se pueda escoger un diseño que no sea FNBC.

7.24. Dese una descomposición en 3FN de reunión sin pérdida que conserve las dependencias del esquema *R* del Ejercicio 7.2.

7.25. Sea un atributo *primo* uno que aparece como mínimo en una clave candidata. Sean α y β conjuntos de atributos tales que se cumple $\alpha \rightarrow \beta$, pero no se cumple $\beta \rightarrow \alpha$. Sea *A* un atributo que no esté en α ni en β y para el que se cumple $\beta \rightarrow \alpha$. Se dice que *A* es *dependiente de manera transitiva* de α. Se puede reformular la definición de 3FN de la manera siguiente: un esquema de relación *R* está en la 3FN con respecto a un conjunto *F* de dependencias funcionales si no hay atributos no primos *A* en *R* para los cuales *A* sea dependiente de manera transitiva de una clave de *R*.

Demuéstrese que esta nueva definición es equivalente a la original.

7.26. Una dependencia funcional $\alpha \rightarrow \beta$ se denomina **dependencia parcial** si hay un subconjunto adecuado γ de α tal que $\gamma \rightarrow \beta$. Se dice que β es *parcialmente dependiente* de α. Un esquema de relación *R* está en la **segunda forma normal** (2FN) si cada atributo *A* de *R* cumple uno de los criterios siguientes:

- Aparece en una clave candidata.
- No es parcialmente dependiente de una clave candidata.

Demuéstrese que cada esquema 3FN se halla en 2FN. (*Sugerencia*: demuéstrese que cada dependencia parcial es una dependencia transitiva.)

7.27. Dados los tres objetivos del diseño de bases de datos relacionales, indíquese si hay alguna razón para diseñar un esquema de base de datos que se halle en 2FN, pero que no se halle en una forma normal de orden

```
resultado := 2
I* cuentadf es un array cuyo elemento iésimo contiene el número de atributos del lado izquierdo de la iésima DF que todavía no
   se sabe que estén en α⁺ */
for i := 1 to |F| do
    begin
        Supongamos que β → γ denota la iésima DF;
        cuentadf [i] :=|β|;
    end
/* aparece es un array con una entrada por cada atributo. La entrada del atributo A es una lista de enteros. Cada entero i de la lis-
   ta indica que A aparece en el sado izquierdo de la i-ésima DF */
for each atributo A do
    begin
        aparece [A] := NIL;
        for i := 1 to |F| do
            begin
                Supongamos que β → γ denota la iésima DF;
                if A ∈ β then añadir i a aparece [A];
            end
    end
agregar (α);
return (resultado);

procedure agregar (α);
for each atributo A de α do
    begin
        if A ∉ resultado then
            begin
                resultado := resultado ∪ {A};
                for each elemento i de aparece[A] do
                    begin
                        cuentadf [i] :=cuentadf [i] – 1;
                        if cuentadf [i] := 0 then
                            begin
                                supongamos que β → γ denota la i-ésima DF;
                                agregar (γ);
                            end
                    end
            end
    end
```

FIGURA 7.22. Un algoritmo para calcular α^+.

superior. (Véase el Ejercicio 7.26 para obtener la definición de 2FN.)

7.28. Dese un ejemplo de esquema de relación *R* y un conjunto de dependencias tales que *R* se halle en FNBC, pero que no esté en 4FN.

7.29. Explíquese el motivo de que 4FN sea una forma normal más deseable que FNBC.

7.30. Explíquese el modo en que pueden aparecer las tuplas colgantes. Explíquense los problemas que pueden provocar.

NOTAS BIBLIOGRÁFICAS

El primer estudio de la teoría del diseño de bases de datos relacionales apareció en un documento pionero de Codd [1970]. En ese documento Codd introducía también las dependencias funcionales y la primera, la segunda y la tercera formas normales.

Los axiomas de Armstrong se introdujeron en Armstrong [1974]. Ullman [1988] es una fuente fácilmente accesible de las pruebas de seguridad y de completitud de los axiomas de Armstrong. Ullman [1988] ofrece también un algoritmo para la comprobación de la descomposición de reunión sin pérdida para las descomposiciones generales (no binarias), y muchos otros algoritmos, teoremas y demostraciones relativos a la teoría de las dependencias. Maier [1983] estudia la teoría de las dependencias funcionales. Graham et al. [1986] estudia los aspectos formales del concepto de relación legal.

FNBC se introdujo en Codd [1972]. Las ventajas de FNBC se estudian en Bernstein et al. [1980a]. En Tsou y Fischer [1982] aparece un algoritmo polinómico en el tiempo para la descomposición en FNBC, y también

puede hallarse en Ullman [1988]. Biskup et al. [1979] ofrecen el algoritmo que se ha utilizado aquí para buscar una descomposición en 3FN de reunión sin pérdida que conserve las dependencias. Los resultados fundamentales de la propiedad de reunión sin pérdida aparecen en Aho et al. [1979a].

Las dependencias multivaloradas se estudian en Zaniolo [1976]. Beeri et al. [1977] dan un conjunto de axiomas para las dependencias multivaloradas y demuestran que los axiomas de los autores son seguros y completos. La axiomatización de este libro se basa en la suya. Los conceptos de 4FN, FNRP y FNDC proceden de Fagin [1977], Fagin [1979] y Fagin [1981], respectivamente.

Maier [1983] presenta con detalle la teoría del diseño de bases de datos relacionales. Ullman [1988] y Abiteboul et al. [1995] presentan un tratamiento más teórico de muchas de las dependencias y formas normales aquí presentadas. Véanse las notas bibliográficas del Apéndice C para obtener más referencias de literatura sobre normalización.



BASES DE DATOS BASADAS EN OBJETOS Y XML

Varias áreas de aplicación de los sistemas de bases de datos están limitadas por las restricciones del modelo de datos relacional. En consecuencia, los investigadores han desarrollado varios modelos de datos para abordar estos campos de aplicación.

En esta parte se estudiará el modelo de datos orientado a objetos y el modelo de datos relacional orientado a objetos. Además, se estudiará XML, un lenguaje que puede representar datos que están menos estructurados que los de otros modelos de datos.

El modelo de datos orientado a objetos, descrito en el Capítulo 8, está basado en el paradigma de los lenguajes de programación orientados a objetos, que en este momento tienen un gran uso. La herencia, la identidad de objetos, y el encapsulamiento (información oculta), con métodos para proporcionar una interfaz a los objetos, están entre los conceptos clave de la programación orientada a objetos que han encontrado aplicaciones en los modelos de datos. Los modelos de datos orientados a objetos también soportan un rico sistema de tipos, incluyendo tipos colección y estructurados. Mientras la herencia y, hasta cierto punto, los tipos complejos están también presentes en el modelo E-R, el encapsulamiento y la identidad de objetos distinguen el modelo de datos orientado a objetos frente al modelo E-R.

Los modelos relacionales orientados a objetos, descritos en el Capítulo 9, combinan rasgos de los modelos relacionales y de los modelos orientados a objetos. Este modelo proporciona el sistema de tipos enriquecido de las bases de datos orientadas a objetos, combinado con relaciones como las básicas para el almacenamiento de datos. Así, se aplica la herencia a las relaciones, no sólo a los tipos. El modelo de datos relacional orientado a objetos proporciona una suave migración desde las bases de datos relacionales que resulta atractiva para los vendedores de bases de datos. En consecuencia, la norma SQL:1999 incluye una serie de rasgos orientados a objetos dentro de su sistema de tipos, mientras continúa el uso de modelos relacionales como el modelo subyacente.

El lenguaje XML fue designado inicialmente como una forma de añadir mayor cantidad de información a los documentos de texto, pero ha llegado a ser importante por sus aplicaciones en intercambio de datos. XML proporciona una forma para representar datos que tienen estructuras anidadas, y además permite una gran flexibilidad estructurando datos, lo cual es importante para ciertos tipos de datos no tradicionales. En el Capítulo 10 se describe el lenguaje XML, y a continuación se presentan diferentes consultas sobre los datos representados en XML y la transformación de datos XML desde una forma a otra.

CAPÍTULO

8 BASES DE DATOS ORIENTADAS A OBJETOS

De igual forma que los sistemas de bases de datos fueron aplicados a rangos más amplios de aplicaciones, incluyendo, por ejemplo, diseño asistido por computadora, las limitaciones impuestas por el modelo relacional han surgido como obstáculos. En consecuencia, los investigadores de bases de datos inventaron nuevos modelos de datos que resuelven las limitaciones del modelo de datos relacional. Este capítulo se concentra en uno de ellos, el modelo orientado a objetos, que está basado en el paradigma de la programación orientada a objetos. El Capítulo 9 trata otro modelo de datos, el modelo de datos relacional orientado a objetos, que combina las características de los modelos orientado a objetos y relacional.

El enfoque orientado a objetos para la programación fue introducida por primera vez con el lenguaje Simula 67, que se diseñó para la programación de simulaciones. Smalltalk fue uno de los primeros lenguajes de programación orientada a objetos para aplicaciones generales. Actualmente, los lenguajes C++ y Java son los lenguajes de programación orientada a objetos más usados.

En este capítulo se introducen los conceptos de programación orientada a objetos y a continuación se considera el uso de estos conceptos en sistemas de bases de datos.

8.1. NECESIDADES DE LOS TIPOS DE DATOS COMPLEJOS

Las aplicaciones de bases de datos tradicionales consisten en tareas de procesamiento de datos, tales como la banca y la gestión de nóminas. Dichas aplicaciones presentan conceptualmente tipos de datos simples. Los elementos de datos básicos son registros bastante pequeños y cuyos campos son atómicos, es decir, no contienen estructuras adicionales y en los que se cumple la primera forma normal (véase el Capítulo 7).

En los últimos años, la demanda ha incrementado las formas de abordar los tipos de datos más complejos. Considérense, por ejemplo, un conjunto de direcciones. Mientras una dirección completa puede ser vista como un elemento de datos atómico de tipo cadena de caracteres, esta forma de verlo escondería detalles como la calle, la población, la provincia, y el código postal que podrían ser interesantes para las consultas. Por otra parte, si una dirección se representa dividiéndola en componentes (calle, población, provincia y código postal) las consultas escritas serían más complicadas, pues tendrían que mencionar cada campo. Una alternativa mejor es permitir tipos de datos estructurados, que admiten un tipo dirección con subpartes *calle*, *población*, *provincia* y *código postal*.

Otro ejemplo: considérense los atributos multivalorados del modelo E-R. Tales atributos son naturales, por ejemplo, para la representación de números de teléfono, ya que las personas pueden tener más de un teléfono. La alternativa de normalización con la creación de una nueva relación es costosa y artificial para este ejemplo.

También como ejemplo, considérese una base de datos para diseño de circuitos electrónicos asistido por computadora. Un circuito usa muchos componentes, de diferentes tipos, y éstos tienen que hacer referencia a otros componentes a los que están conectados. Todos los componentes de un mismo tipo presentan las mismas propiedades. El modelado del circuito en la base de datos es más simple si se puede visualizar cada componente en el diseño como un *ejemplar* de tipo componente, y dar a cada ejemplar un *identificador* único. Los componentes del circuito pueden entonces referirse a otros componentes a través de su identificador único.

Supóngase ahora que unos ingenieros deseen determinar el consumo de energía del circuito completo. Una buena forma de hacer esto es ver cada componente como un *objeto* proporcionando una función *usodeenergía()* que dice cuánta energía usan las componentes. La función que computa toda la energía usada no necesita conocer nada sobre el interior de los componentes; sólo es necesario invocar a la función *usodeenergía()* en cada componente y añadirlo al resultado.

Se examinará la cuestión planteada con más detalle en el resto de este capítulo.

8.2. EL MODELO DE DATOS ORIENTADO A OBJETOS

En esta sección se presentan los conceptos principales del modelo de datos orientado a objetos: la estructura de éstos y las nociones de clases, herencia e identidad de objetos.

8.2.1. Estructura de los objetos

Hablando en general, los *objetos* se corresponden con las entidades del modelo E-R. El paradigma orientado a objetos está basado en el *encapsulamiento* de los datos y del código relacionados con cada objeto en una sola unidad cuyo contenido no es visible desde el exterior. Conceptualmente, todas las interacciones entre cada objeto y el resto del sistema se realizan mediante mensajes. Por tanto, la interfaz entre cada objeto y el resto del sistema se define mediante un conjunto de *mensajes* permitidos.

En general, cada objeto está asociado con

- Un conjunto de *variables* que contiene los datos del objeto; las variables se corresponden con los atributos del modelo E-R.
- Un conjunto de *mensajes* a los que responde; cada mensaje puede no tener *parámetros*, tener uno o varios.
- Un conjunto de *métodos*, cada uno de los cuales es código que implementa un mensaje; el método devuelve un valor como *respuesta* al mensaje.

El término *mensaje* en un entorno orientado a objetos no implica el uso de mensajes físicos en redes informáticas. Por el contrario hace referencia al intercambio de solicitudes entre los objetos independientemente de los detalles concretos de su implementación. Se utiliza a veces la expresión **invocar a un método** para denotar el hecho de enviar un mensaje a un objeto y la ejecución del método correspondiente.

Se puede explicar la razón del uso de este enfoque considerando las entidades *empleado* de una base de datos bancaria. Supóngase que el sueldo anual de cada empleado se calcula de manera diferente para los distintos empleados. Por ejemplo, puede que los jefes obtengan una prima en función de los resultados del banco, mientras que los cajeros reciben una prima en función de las horas que hayan trabajado. Se puede (en teoría) encapsular el código para calcular su sueldo con cada empleado en forma de método que se ejecute en respuesta a un mensaje de *sueldo-anual*.

Todos los objetos *empleado* responden al mensaje *sueldo-anual*, pero lo hacen de manera diferente. Al encapsular con el objeto *empleado* la información sobre el cálculo de su sueldo anual, todos los objetos *empleado* presentan la misma interfaz. Dado que la única interfaz externa presentada por cada objeto es el conjunto de mensajes a los que responde, resulta posible modificar las definiciones de los métodos y de las variables sin afectar al resto del sistema. La posibilidad de modificar la definición de un objeto sin afectar al resto del sistema se considera una de las mayores ventajas del paradigma de la programación orientada a objetos.

Los métodos de cada objeto pueden clasificarse como *sólo de lectura* o *de actualización*. Los métodos sólo de lectura no afectan al valor de las variables de los objetos, mientras que los métodos de actualización sí pueden modificarlo. Los mensajes a los que responde cada objeto pueden clasificarse de manera parecida como sólo de lectura o de actualización, según el método que los implemente.

Los atributos derivados de las entidades del modelo E-R pueden expresarse en el modelo orientado a objetos como mensajes sólo de lectura. Por ejemplo, el atributo derivado *antigüedad* de una entidad *empleado* puede expresarse como el mensaje *antigüedad* de un objeto *empleado*. El método que implemente los mensajes, puede determinar la antigüedad restando la *fecha-alta* del empleado de la fecha actual.

Hablando con rigor, en el modelo orientado a objetos hay que expresar cada atributo de las entidades como una variable y un par de mensajes del objeto correspondiente. La variable se utiliza para guardar el valor del atributo, uno de los mensajes se utiliza para leer el valor del atributo y el otro mensaje se utiliza para actualizar ese valor. Por ejemplo, el atributo *dirección* de la entidad *empleado* puede representarse mediante:

- Una variable *dirección*.
- Un mensaje *obtener-dirección* cuya respuesta sea la dirección.
- Un mensaje *establecer-dirección*, que necesita un parámetro *nueva-dirección*, para actualizar la dirección.

Sin embargo, en aras de la sencillez, muchos modelos orientados a objetos permiten que las variables se lean o se actualicen de manera directa, sin necesidad de definir los mensajes para ello.

8.2.2. Clases de objetos

Generalmente, en una base de datos hay muchos objetos similares. Por *similar* se entiende que responden a los mismos mensajes, utilizan los mismos métodos y tienen variables del mismo nombre y del mismo tipo. Sería un derroche definir por separado cada uno de estos objetos. Por tanto, los objetos parecidos se agrupan para formar una **clase**. Cada uno de estos objetos se denomina **ejemplar** de su clase. Todos los objetos de una

clase comparten una definición común, pese a que se diferencien en los valores asignados a las variables.

El concepto de clase del modelo orientado a objetos se corresponde con el concepto de entidad del modelo E-R. Algunos ejemplos de clases en la base de datos bancaria son los empleados, los clientes, las cuentas y los préstamos.

La Figura 8.1 define la clase empleado en pseudocódigo. La definición muestra las variables y los mensajes a los que responden los objetos de la clase. En esta definición, cada objeto de la clase *empleado* contiene las variables *nombre* y *dirección*, ambas cadenas de caracteres; *fecha-alta*, que es una fecha, y *sueldo*, que es un entero. Cada objeto responde a los cinco mensajes mostrados, llamados *sueldo-anual*, *obtener-nombre*, *obtener-dirección*, *establecer-dirección* y *antigüedad*. El nombre de tipo que precede a cada mensaje indica el tipo de la respuesta al mismo. Obsérvese que el mensaje *establecer-dirección* utiliza el parámetro *nueva-dirección* que especifica el nuevo valor de la calle. Aunque no lo hemos mostrado aquí, la clase *empleado* soportaría también mensajes que establecen el nombre, el sueldo y la fecha de alta.

Los métodos para el manejo de mensajes suelen definirse separados de la definición de clases. Los métodos *obtener-dirección*() y *establecer-dirección*() estarían definidos, por ejemplo, por el pseudocódigo:

```
string obtener-dirección() {
    return dirección;
}
int establecer-dirección(string nueva-dirección) {
    dirección = nueva-dirección;
}
```

mientras que el método *antigüedad*() se definiría:

```
int antigüedad(){
    return today() – fecha-alta;
}
```

Aquí, asumimos que la función *today*() es una función que devuelve la fecha actual, y el «–» opera con ellas devolviendo el intervalo entre las dos fechas.

```
class empleado {
    /* Variables */
        string  nombre;
        string  dirección;
        date    fecha-alta;
        int     sueldo;
/* Mensajes */
        int     sueldo-anual ();
        string  obtener-nombre ();
        string  obtener-dirección ();
        int     establecer-dirección (string nueva-dirección);
        int     antigüedad();
};
```

FIGURA 8.1. Definición de la clase *empleado*.

El concepto de clases es parecido al concepto de los tipos abstractos de datos. Sin embargo, hay varios aspectos adicionales en el concepto de clase respecto al de tipos abstractos de datos. Para representar estas propiedades adicionales, cada clase se trata como si fuera un objeto. Un objeto clase incluye

- Una variable de tipo conjunto cuyo valor es el conjunto de todos los objetos que son ejemplares de la clase.
- La implementación de un método para el mensaje *nuevo*, que crea un nuevo ejemplar de la clase.

Se retomarán estas características en el Apartado 8.5.2.

8.2.3. Herencia

Los esquemas de las bases de datos orientadas a objetos suelen necesitar gran número de clases. Frecuentemente, sin embargo, varias de las clases son parecidas entre sí. Por ejemplo, supóngase que se tiene una base de datos orientada a objetos en la aplicación bancaria. Cabe esperar que la clase de los clientes del banco sea parecida a la clase de los empleados en que ambas definan variables para *nombre*, *dirección*, etcétera. Sin embargo, hay algunas variables específicas de los empleados (*sueldo*, por ejemplo) y otras específicas de los clientes (*interés-préstamo*, por ejemplo). Sería conveniente definir una representación de las variables comunes en un solo lugar. Esto sólo puede hacerse si se combinan los empleados y los clientes en una sola clase.

Para permitir la representación directa de los parecidos entre las clases hay que ubicarlas en una jerarquía de especializaciones (la relación «ES») como la definida en el Capítulo 2 para el modelo entidad-relación. Por ejemplo, se puede decir que *empleado* es una especialización de *persona*, dado que el conjunto de los empleados es un subconjunto del conjunto de personas. Es decir, todos los empleados son personas. De manera parecida, *cliente* es una especialización de *persona*.

El concepto de **jerarquía de clases** es parecido al de especialización del modelo entidad-relación. Los empleados y los clientes pueden representarse mediante clases que son especializaciones de la clase *persona*. Las variables y los métodos específicos de los empleados se asocian con la clase *empleado*. Las variables y los métodos específicos de los clientes se asocian con la clase *cliente*. Las variables y los métodos que se aplican tanto a empleados como a clientes se asocian con la clase *persona*.

En la Figura 8.2 se muestra un diagrama E-R con una jerarquía de especializaciones que representa a las personas implicadas en la operación del ejemplo bancario. En la Figura 8.3 se muestra la jerarquía de clases correspondiente. Las clases mostradas en la jerarquía pueden definirse en pseudocódigo como se muestra en la Figura 8.4. Por brevedad no se presentan todos los mensajes y métodos asociados con estas clases, aunque

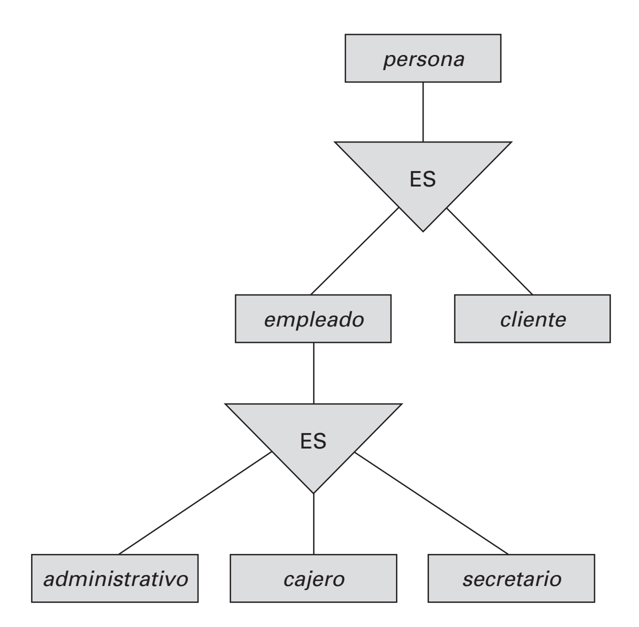


FIGURA 8.2. Jerarquía de especializaciones para el ejemplo bancario.

se deben incluir en una definición completa de la base de datos bancaria.

La palabra clave **isa** (es) se utiliza para indicar que una clase es una especialización de otra. Las especializaciones de las clases se denominan **subclases.** Por tanto, por ejemplo, *empleado* es una subclase de *persona* y *cajero* es una subclase de *empleado*. Análogamente, *empleado* es una **superclase** de *cajero* y *persona* es una superclase de *empleado*.

Los objetos que representan a administrativos contienen todas las variables de la clase *administrativo*; además, los objetos que representan a administrativos contienen también todas las variables de las clases *empleado* y *persona*. Esto se debe a que los administrativos se definen como empleados y, como los empleados a su vez se definen como personas, se puede deducir que los administrativos son también personas. Esta propiedad de que los objetos de una clase contengan las variables definidas en sus superclases se denomina **herencia** de las variables.

Los mensajes y los métodos se heredan de modo idéntico a las variables. Una ventaja importante de la

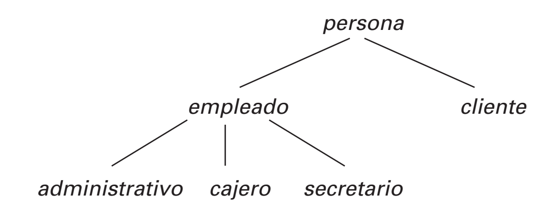


FIGURA 8.3. Jerarquía de clases correspondiente a la Figura 8.2.

```
class persona {
    string  nombre;
    string  dirección;
    string  obtener-nombre();
    string  obtener-dirección();
    int     establecer-dirección(string nueva-dirección);
};
class cliente isa persona {
    int     interés-préstamo;
};
class empleado isa persona {
    date    fecha-alta;
    int     sueldo;
    int     sueldo-anual();
    int     antigüedad();
};
class administrativo isa empleado {
    int     número-despacho;
    int     número-cuenta-corriente;
};
class cajero isa empleado {
    int     horas-semana;
    int     número-ventanilla;
};
class secretario isa empleado {
    int     horas-semana;
    string  jefe;
};
```

FIGURA 8.4. Definición en pseudocódigo de una jerarquía de clases.

herencia en los sistemas orientados a objetos es el concepto de **posibilidad de sustitución**: cualquier método de una clase dada —por ejemplo, *A* (o una función que utilice un objeto de la clase *A* como argumento)— puede ser llamado de igual modo con cualquier objeto perteneciente a cualquier subclase *B* de *A*. Esta característica lleva a la reutilización del código, dado que no hace falta volver a escribir los mensajes, métodos y funciones para los objetos de la clase B. Por ejemplo, si se define el mensaje *obtener-nombre* para la clase *persona*, se puede llamar con un objeto *persona* o con cualquier objeto perteneciente a cualquiera de las subclases de *persona*, como *cliente* o *administrativo*.

En el Apartado 8.2.2 se observó que cada clase es un objeto por sí misma y que ese objeto incluye una variable que contiene el conjunto de todos los ejemplares de la clase. Resulta sencillo determinar los objetos que se hallan asociados con las clases en las hojas de la jerarquía. Por ejemplo, se asocia con la clase *cliente* el conjunto de los clientes del banco. Para las clases que no son hojas, sin embargo, el problema resulta más complejo. En la jerarquía de la Figura 8.3 hay dos maneras posibles de asociar los objetos con las clases:

1. Se pueden asociar todos los objetos empleado, incluyendo aquellos que sean elementos de *administrativo*, de *cajero* o de *secretario*, con la clase *empleado*.

2. Sólo se pueden asociar con la clase *empleado* los objetos empleado que no sean elementos de *administrativo* ni de *cajero* ni de *secretario*.

En los sistemas orientados a objetos generalmente se escoge la segunda opción. En este caso resulta posible determinar el conjunto de objetos empleado tomando la unión de los objetos asociados con todas las subclases de *empleado*. La mayor parte de los sistemas orientados a objetos permiten que la especialización sea parcial; es decir, permiten objetos que pertenezcan a una clase, como *empleado*, pero que no pertenezcan a ninguna de las subclases de la misma.

8.2.4. Herencia múltiple

En la mayor parte de los casos una **organización de clases con estructura de árbol** resulta adecuada para describir las aplicaciones; en la organización con estructura de árbol, cada clase puede tener a lo sumo una superclase. Sin embargo, hay situaciones que no pueden representarse bien en una jerarquía de clases con estructura de árbol.

La **herencia múltiple** permite a las clases heredar variables y métodos de múltiples superclases. La relación entre clases y subclases se representa mediante un ***grafo acíclico dirigido*** (GAD) en el que las clases pueden tener más de una superclase.

Por ejemplo, supóngase que los empleados pueden ser contratados por horas (para un periodo limitado) o bien a tiempo completo. Se pueden crear las subclases *por-horas* y *a-tiempo-completo* de la clase *empleado*. La subclase *por-horas* tendría un atributo *último-día* que especifica cuándo concluye el periodo de empleo. La subclase *a-tiempo-completo* puede tener un método para el cálculo de las contribuciones al plan de pensiones de la compañía, que no es aplicable a los empleados por horas.

La clasificación expuesta de empleados como temporal y a tiempo completo es independiente de la clasificación basada en el trabajo que ellos realizan, es decir, administrativo, cajero, o secretario. Se podrían tener administrativos a tiempo completo, administrativos por horas, cajeros a tiempo completo y cajeros por horas. Usando la herencia múltiple, simplemente se crea una nueva clase, tal como *cajero-por-horas*, que es una subclase de *por-horas* y de *cajero*, y *cajero-a-tiempo-completo*, que es una subclase de *a-tiempo-completo* y de *cajero*. La combinación que no puede ocurrir en la vida real no necesita ser creada; por ejemplo, si todos los administrativos son a tiempo completo, no es necesario crear una clase de administrativos por horas. La jerarquía de clases resultantes aparece en la Figura 8.5.

Gracias a la herencia múltiple, los atributos y los métodos concretos de los empleados por horas y a tiempo completo se especifican sólo una vez y los heredan todas las subclases. En cambio, sin la herencia múltiple, se debería, por ejemplo, definir *administrativo-por-horas* como una subclase de *administrativo*, y *cajero-por-horas* como una subclase de *cajero*. Los atributos y métodos específicos para empleados por horas tendrían que ser repetidos en cada una de las subclases expuestas.

Otro ejemplo de herencia múltiple es considerar una base de datos universitaria, donde una persona puede ser estudiante o profesor. La base de datos universitaria puede tener clases *estudiante* y *profesor*, que son subclases de *persona,* para modelar esta situación. Consideremos ahora también una categoría de estudiantes que trabajan como ayudantes de profesor; para modelar esta situación se puede crear una clase *ayudante-pro-*

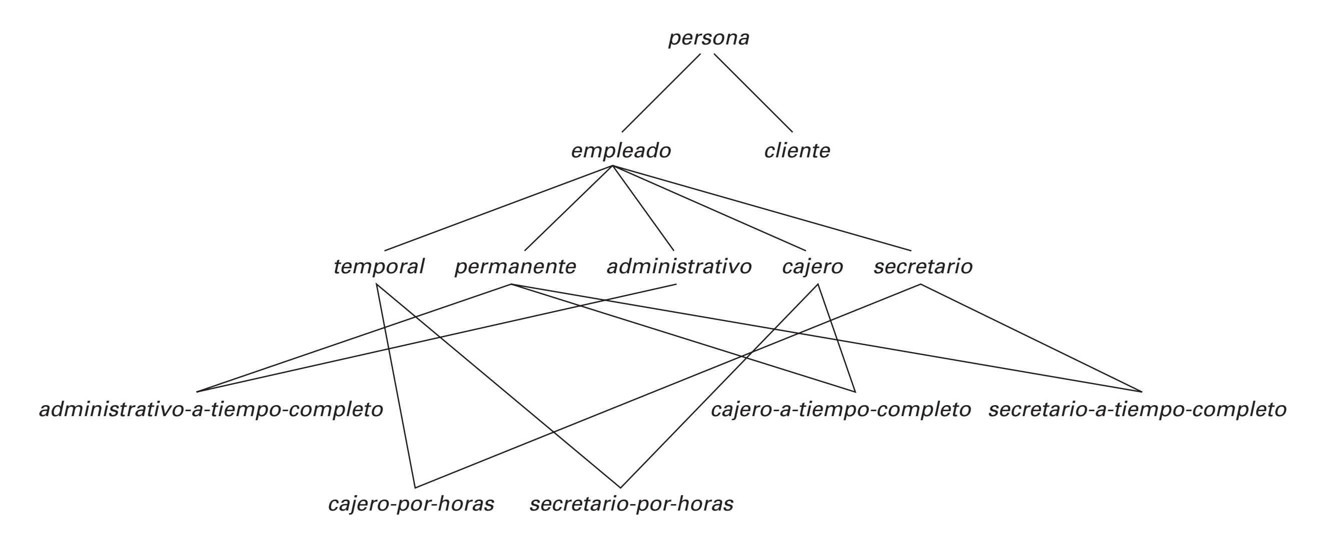


FIGURA 8.5. GAD de clases del ejemplo bancario.

fesor como una subclase tanto de *estudiante* como de *profesor.*

Cuando se utiliza la herencia múltiple aparece una ambigüedad potencial si se puede heredar la misma variable o el mismo método de más de una superclase. Por ejemplo, la clase *estudiante* puede tener una variable *dept* que identifica el departamento del estudiante y la clase *profesor* puede tener análogamente una variable *dept* que identifica el departamento del profesor. La clase *ayudante-profesor* hereda ambas definiciones de la variable *dept*; así el uso de la variable *dept* en el contexto de ayudantes de profesor se vuelve ambigua.

El resultado de usar *dept* varía dependiendo de la implementación particular del modelo orientado a objetos. Por ejemplo, la implementación puede manejar *dept* de una de estas formas:

- Incluir las dos variables, dándoles el nombre *estudiante.dept* y *profesor.dept*.
- Escoger una de las dos según el orden en que se crearon las clases *estudiante* y *profesor*.
- Hacer que el usuario escoja de manera explícita una de las opciones en la definición de la clase *ayudante-profesor.*
- Tratar la situación como si fuera un error.

Ninguna de estas soluciones se ha aceptado como óptima y los diferentes sistemas implementan opciones diferentes.

No todos los casos de herencia múltiple llevan a ambigüedad. Por ejemplo, la variable *sueldo* está definida en la clase *empleados*; todas las clases *administrativo, cajero* y *secretario* igual que *por-horas* y *a-tiempo-completo* heredan sueldo de *empleado*. Dado que estas tres clases comparten la misma definición de *sueldo,* no surge ambigüedad de la herencia de *sueldo* por *cajero-a-tiempo-completo*, *administrativo-a-tiempo-completo*, y así sucesivamente.

Considérese de nuevo la base de datos universitaria. La base de datos puede tener varias subclases de *persona,* incluyendo *estudiante* y *profesor* como se vio anteriormente en este apartado, y además subclases como *miembro-consejo* (esto es, un miembro de un consejo de estudiantes y profesores que se encarga de los asuntos de los estudiantes). Un objeto puede pertenecer a varias de estas categorías de manera simultánea y cada una de éstas se denomina *papel*. La noción de papel aquí es similar a los papeles usados para la autorización en SQL (véase el Capítulo 6).

Muchos lenguajes de programación orientados a objetos insisten en que un objeto debe tener una **clase más específica**, es decir, si un objeto pertenece a muchas clases, se debe pertenecer a una clase que es (directa o indirectamente) subclase de todas las otras clases a las que el objeto pertenece. Por ejemplo, si un objeto pertenece a las clases *persona* y *profesor*, tiene que pertenecer a alguna clase que es subclase de ambas de estas clases. Se tiene que usar entonces herencia múltiple para crear todas las subclases requeridas, tales como *ayudante-profesor* (que es una subclase de *estudiante* y *profesor*), *estudiante-miembro-consejo, ayudante-profesor-miembro-consejo*, etcétera, para modelar la posibilidad de que un objeto tenga varios papeles de manera simultánea.

Claramente, la creación de muchas subclases es bastante incómoda, y los sistemas que no tienen necesidad de objetos con una clase más específica son preferibles para modelar papeles. En el Capítulo 9 se discute una ampliación de SQL que presenta una manera alternativa de modelar papeles.

8.2.5. Identidad de los objetos

Los objetos de las bases de datos orientadas a objetos suelen corresponder a entidades del sistema modelado por la base de datos. Las entidades conservan su identidad aunque algunas de sus propiedades cambien con el tiempo. De manera parecida, los objetos conservan su identidad aunque los valores de las variables o las definiciones de los métodos cambien total o parcialmente con el tiempo. Este concepto de identidad no se aplica a las tuplas de las bases de datos relacionales. En los sistemas relacionales las tuplas de una relación sólo se distinguen por los valores que contienen.

La identidad de los objetos es un concepto de identidad más potente que el que suele hallarse en los lenguajes de programación o en los modelos de datos que no se basan en la programación orientada a objetos. A continuación se ilustran varios ejemplos de identidad.

- **Valor.** Se utiliza un valor de datos como identidad. Esta forma de identidad se utiliza en los sistemas relacionales. Por ejemplo, el valor de la clave primaria de una tupla identifica a la tupla.
- **Nombre.** Se utiliza como identidad un nombre proporcionado por el usuario. Esta forma de identidad suele utilizarse para los archivos en los sistemas de archivos. Cada archivo recibe un nombre que lo identifica de manera unívoca, independientemente de su contenido.
- **Incorporada.** Se incluye el concepto de identidad en el modelo de datos o en el lenguaje de programación y no hace falta que el usuario proporcione ningún identificador. Esta forma de identidad se utiliza en los sistemas orientados a objetos. Cada objeto recibe del sistema de manera automática un identificador en el momento en que se crea.

La identidad de los objetos es una noción conceptual; los sistemas reales necesitan un mecanismo físico que *identifique* los objetos de manera unívoca. Para los seres humanos se suelen utilizar como identificadores

los nombres, junto con otra información como la fecha y el lugar de nacimiento. Los sistemas orientados a objetos proporcionan el concepto de **identificador del objeto** para identificar a los objetos. Los identificadores de los objetos son **únicos;** es decir, cada objeto tiene un solo identificador y no hay dos objetos que tengan el mismo identificador. Los identificadores de los objetos no tienen por qué estar en una forma con la que los seres humanos se encuentren cómodos; pueden ser números grandes, por ejemplo. La posibilidad de guardar el identificador de un objeto como un campo de otro objeto es más importante que tener un nombre que resulte fácil de recordar.

Como ejemplo del uso de los identificadores de los objetos, uno de los atributos de los objetos *persona* puede ser el atributo *cónyuge*, que es en realidad un identificador del objeto *persona* correspondiente al cónyuge de la primera persona. Por tanto, el objeto **persona** puede guardar una *referencia* del objeto que representa al cónyuge de esa persona.

Generalmente el identificador lo genera el sistema de manera automática. Por el contrario, en las bases de datos relacionales, el campo *cónyuge* de la relación *persona* será generalmente un identificador unívoco (quizás un número de DNI) del cónyuge de esa persona generado de manera externa al sistema de bases de datos. Hay muchas situaciones en las que hacer que el sistema genere identificadores de manera automática resulta una ventaja, dado que libera a los seres humanos de llevar a cabo esa tarea. Sin embargo, esta posibilidad debe utilizarse con precaución. Los identificadores generados por el sistema suelen ser específicos del mismo y, si se desplazan los datos a un sistema de bases de datos diferente, hay que traducirlos. Además, los identificadores generados por el sistema pueden resultar redundantes si las entidades que se modelan ya disponen de identificadores unívocos externos al sistema. Por ejemplo, en España los números del DNI suelen utilizarse como identificadores unívocos de las personas.

8.2.6. Continentes de objetos

Se pueden utilizar las referencias entre objetos para modelar diferentes conceptos del mundo real. Uno de estos objetos es el de *continentes de objetos*. Para ilustrar los continentes de objetos considérese la base de datos simplificada para diseño de bicicletas de la Figura 8.6. Cada diseño de bicicletas contiene las ruedas, el cuadro, los frenos y los cambios. Las ruedas, a su vez, contienen la llanta, un conjunto de radios y el neumático. Todos los componentes del diseño pueden modelarse como objetos y los continentes de los componentes puede modelarse como continentes de objetos.

Los objetos que contienen a otros objetos se denominan **objetos complejos** o **compuestos.** Puede haber varios niveles de continentes, como se muestra en la Figura 8.6. Esta situación crea entre los objetos una **jerarquía de continentes**.

La jerarquía de la Figura 8.6 muestra la relación de continentes entre los objetos de manera esquemática dando los nombres de las clases en vez de los de los objetos concretos. Los enlaces entre las clases deben interpretarse como «**es parte de**» en lugar de la interpretación «**es**» de los enlaces de una jerarquía de herencias.

Las variables de las diferentes clases no se muestran en la figura. Considérese la clase *bicicleta*. Puede incluir variables como *marca* que contengan datos descriptivos de los elementos de la clase *bicicleta*. Además, la clase *bicicleta* incluye las variables que incluyen referencias o conjuntos de referencias a los objetos de las clases *ruedas*, *frenos*, *cambio* y *cuadro*.

El concepto de continente es importante en los sistemas orientados a objetos porque permite que los diferentes usuarios examinen los datos con diferente detalle. Los diseñadores de ruedas pueden concentrarse en los elementos de la clase *ruedas* sin tener que preocuparse mucho, si no tienen que preocuparse en absoluto, de los objetos de las clases *cambio* o *frenos*. Los empleados de márketing que intenten fijar el precio de toda la bicicleta pueden hacer referencia a todos los datos correspondientes a la bicicleta haciendo referencia al elemento adecuado de la clase *bicicleta*. La jerarquía de continentes se utiliza para buscar todos los objetos contenidos en los objetos *bicicleta*.

En ciertas aplicaciones un objeto puede estar incluido en varios objetos. En esos casos la relación de continentes se representa mediante un GAD en lugar de mediante una jerarquía.

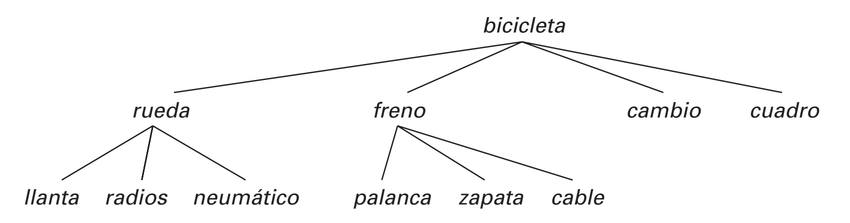


FIGURA 8.6. Jerarquía de continentes de la base de datos para el diseño de bicicletas.

8.3. LENGUAJES ORIENTADOS A OBJETOS

Hasta el momento se han abordado los conceptos básicos de la programación orientada a objetos en un nivel abstracto. Para poder utilizarlos en la práctica en un sistema de bases de datos hay que expresarlos en algún lenguaje. Esta expresión puede realizarse de dos maneras:

1. Los conceptos de la programación orientada a objetos se utilizan simplemente como herramientas de diseño y se codifican, por ejemplo, en una base de datos relacional. Se sigue este enfoque cuando se utilizan los diagramas entidad-relación para modelar los datos y luego se convierten de manera manual en un conjunto de relaciones.
2. Los conceptos de la programación orientada a objetos se incorporan en un lenguaje que se utiliza para trabajar con la base de datos. Con este enfoque hay varios lenguajes posibles en los que se pueden integrar los conceptos:
   - Una opción es extender un lenguaje para el tratamiento de datos como SQL añadiendo tipos complejos y la programación orientada a objetos. Los sistemas que proporcionan extensiones orientadas a objetos a los sistemas relacionales se denominan sistemas *relacionales orientados a objetos*. Los sistemas relacionales orientados a objetos, y en concreto las extensiones de SQL orientadas a objetos, se describen en el Capítulo 9.
   - Otra opción es tomar un lenguaje de programación orientado a objetos ya existente y extenderlo para que trabaje con las bases de datos. Estos lenguajes se denominan lenguajes de **programación persistente.**

Hay diferentes situaciones en las que resulta más adecuado uno u otro de los enfoques. El segundo enfoque se estudia en los apartados siguientes de este capítulo. La relación entre ambos se discute en el Capítulo 9, después de describir los sistemas relacionales orientados a objetos.

8.4. LENGUAJES DE PROGRAMACIÓN PERSISTENTE

Los lenguajes de las bases de datos se diferencian de los lenguajes de programación tradicionales en que trabajan directamente con datos que son persistentes, es decir, los datos siguen existiendo una vez que el programa que los creó ha concluido. Las relaciones de las bases de datos y las tuplas de las relaciones son ejemplos de datos persistentes. Por el contrario, los únicos datos persistentes con los que los lenguajes de programación tradicionales trabajan directamente son los archivos.

El acceso a las bases de datos es sólo un componente de las aplicaciones del mundo real. Mientras que los lenguajes para el tratamiento de datos como SQL son bastante efectivos en el acceso a los datos, se necesita un lenguaje de programación para implementar otros componentes de las aplicaciones como las interfaces de usuario o la comunicación con otras computadoras. La manera tradicional de realizar las interfaces de las bases de datos con los lenguajes de programación es incorporar SQL dentro del lenguaje de programación.

Los **lenguajes de programación persistente** son lenguajes de programación extendidos con constructoras para el tratamiento de datos persistentes. Los lenguajes de programación persistente pueden distinguirse de los lenguajes con SQL incorporado de al menos dos maneras:

1. En los lenguajes incorporados el sistema de tipos del lenguaje anfitrión suele ser diferente del sistema de tipos del lenguaje para el tratamiento de los datos. Los programadores son responsables de las conversiones de tipos entre el lenguaje anfitrión y SQL. Exigir que los programadores ejecuten esta tarea presenta varios inconvenientes:
   - El código para la conversión entre objetos y tuplas opera fuera del sistema de tipos orientado a objetos y, por tanto, tiene más posibilidades de presentar errores no detectados.
   - La conversión en la base de datos entre el formato orientado a objetos y el formato relacional de las tuplas necesita gran cantidad de código. El código para la conversión de formatos, junto con el código para cargar y descargar los datos de la base de datos, puede suponer un porcentaje significativo del código total necesario para la aplicación.

   Por el contrario, en los lenguajes de programación persistente, el lenguaje de consulta se halla totalmente integrado con el lenguaje anfitrión y ambos comparten el mismo sistema de tipos. Los objetos se pueden crear y guardar en la base de datos sin ningún tipo explícito ni cambios de formato; los cambios de formato necesarios se realizan de manera transparente.

2. Los programadores que utilizan lenguajes de consulta incorporados son responsables de la escritura de código explícito para la búsqueda de los

datos de la base de datos en la memoria. Si se realizan actualizaciones, los programadores deben escribir código de manera explícita para volver a guardar los datos actualizados en la base de datos.

Por el contrario, en los lenguajes de programación persistentes los programadores pueden trabajar con datos persistentes sin tener que escribir de manera explícita código para buscarlos en la memoria o para volver a guardarlos en el disco.

Se han propuesto versiones persistentes de los lenguajes de programación como Pascal. En los últimos años han recibido mucha atención las versiones persistentes de los lenguajes orientados a objetos como C++, Java y Smalltalk. Permiten a los programadores trabajar directamente con los datos desde el lenguaje de programación, sin tener que pasar por un lenguaje para el tratamiento de los datos como SQL. Por tanto, proporcionan una mayor integración de los lenguajes de programación con las bases de datos que, por ejemplo, SQL incorporado.

Sin embargo, los lenguajes de programación persistentes presentan ciertos inconvenientes que hay que tener presentes al decidir si conviene utilizarlos. Dado que los lenguajes de programación suelen ser potentes, resulta relativamente sencillo cometer errores de programación que dañen las bases de datos. La complejidad de los lenguajes hace que la optimización automática de alto nivel, como la reducción de E/S de disco, resulte más difícil. En muchas aplicaciones la posibilidad de las consultas declarativas resulta de gran importancia, pero los lenguajes de programación persistentes no permiten actualmente las consultas declarativas sin que aparezcan problemas de algún tipo.

A continuación se describen varios aspectos que cualquier lenguaje de programación persistente debe abordar y, en apartados posteriores, se describen extensiones de C++ que lo hacen un lenguaje de programación persistente.

8.4.1. Persistencia de los objetos

Los lenguajes de programación orientados a objetos ya poseen un concepto de los mismos, un sistema de tipos para definir sus tipos y constructoras para crearlos. Sin embargo, estos objetos son *transitorios*, desaparecen en cuanto se termina el programa, igual que ocurre con las variables de los programas en Pascal o en C. Si se desea transformar uno de estos lenguajes en un lenguaje para la programación de bases de datos, el primer paso consiste en proporcionar una manera de hacer persistentes a los objetos. Se han propuesto varios enfoques.

- **Persistencia por clases.** El enfoque más sencillo, pero el menos conveniente, consiste en declarar que una clase es persistente. Todos los objetos de la clase son, por tanto, persistentes de manera predeterminada. Todos los objetos de las clases no persistentes son transitorios.

  Este enfoque no es flexible, dado que suele resultar útil disponer en una misma clase tanto de objetos transitorios como de objetos persistentes. En muchos sistemas de bases de datos orientados a objetos la declaración de que una clase es persistente se interpreta como si se afirmara que los objetos de la clase pueden hacerse persistentes en vez de que todos los objetos de la clase son persistentes. Estas clases se podrían haber denominado con más propiedad clases «que pueden ser persistentes».
- **Persistencia por creación.** En este enfoque se introduce una sintaxis nueva para crear los objetos persistentes mediante la extensión de la sintaxis para la creación de los objetos transitorios. Por tanto, los objetos son persistentes o transitorios en función de la manera de crearlos. Este enfoque se sigue en varios sistemas de bases de datos orientados a objetos.
- **Persistencia por marcas.** Una variante del enfoque anterior es marcar los objetos como persistentes después de haberlos creado. Todos los objetos se crean como transitorios, pero, si un objeto tiene que persistir más allá de la ejecución del programa, hay que marcarlo de manera explícita antes de que éste concluya. A diferencia del enfoque anterior, la decisión sobre la persistencia o la transitoriedad se retrasa hasta después de la creación del objeto.
- **Persistencia por alcance.** Uno o varios objetos se declaran objetos persistentes (objetos raíz) de manera explícita. Todos los demás objetos serán persistentes si (y sólo si) son alcanzables desde el objeto raíz mediante una secuencia de una o más referencias.

  Por tanto, todos los objetos a los que se haga referencia desde los objetos persistentes raíz (es decir, aquéllos cuyos identificadores se almacenen en los objetos persistentes raíz) serán persistentes. Pero también lo serán todos los objetos a los que se haga referencia desde ellos, y los objetos a los que estos últimos hagan referencia serán también persistentes, etcétera. Por tanto, los objetos persistentes son exactamente los alcanzables desde una raíz persistente.

  Este esquema tiene la ventaja de que resulta sencillo hacer que sean persistentes estructuras de datos completas con sólo declarar como persistente la raíz de las mismas. Sin embargo, el sistema de bases de datos sufre la carga de tener que seguir las cadenas de referencias para detectar los objetos que son persistentes, y eso puede resultar costoso.

8.4.2. La identidad de los objetos y los punteros

En los lenguajes de programación orientados a objetos que no se han extendido para tratar la persistencia, al

crear objetos se obtienen identificadores de objetos transitorios. Los identificadores de objetos transitorios sólo son válidos mientras se ejecuta el programa que los creó; después de que concluya ese programa el objeto se borra y el identificador pierde su sentido. Cuando se crean objetos persistentes se les asignan identificadores de objetos persistentes.

El concepto de la identidad de los objetos tiene una relación interesante con los punteros de los lenguajes de programación. Una manera sencilla de conseguir una identidad intrínseca es utilizar los punteros a las ubicaciones físicas de almacenamiento. En concreto, en muchos lenguajes orientados a objetos como C++, los identificadores de los objetos son en realidad punteros internos de la memoria.

Sin embargo, la asociación de los objetos con ubicaciones físicas de almacenamiento puede variar con el tiempo. Hay varios grados de permanencia de las identidades:

- **Dentro de los procedimientos.** La identidad sólo persiste durante la ejecución de un único procedimiento. Un ejemplo de la identidad dentro de los procedimientos son las variables locales del interior de los procedimientos.
- **Dentro de los programas.** La identidad sólo persiste durante la ejecución de un único programa o una única consulta. Un ejemplo de la identidad dentro de los programas son las variables globales de los lenguajes de programación. Los punteros de la memoria principal o de la memoria virtual sólo ofrecen identidad dentro de los programas.
- **Entre programas.** La identidad persiste de una ejecución del programa a otra. Los punteros a los datos del sistema de archivos del disco ofrecen identidad entre los programas, pero pueden cambiar si se modifica la manera en que los datos se guardan en el sistema de archivos.
- **Persistente.** La identidad no sólo persiste entre las ejecuciones del programa sino también entre las reorganizaciones estructurales de los datos. Es la forma persistente de la identidad necesaria para los sistemas orientados a objetos.

En las extensiones persistentes de los lenguajes como C++, los identificadores de los objetos para los objetos persistentes se implementan como «punteros persistentes». Un *puntero persistente* es un tipo de puntero que, a diferencia de los punteros internos de la memoria, sigue siendo válido después del final de la ejecución del programa y después de algunas modalidades de reorganización de los datos. Los programadores pueden utilizar los punteros persistentes del mismo modo que utilizan los punteros internos de la memoria en los lenguajes de programación. Conceptualmente los punteros persistentes se pueden considerar como punteros a objetos de la base de datos.

8.4.3. El almacenamiento y el acceso a los objetos persistentes

¿Qué significa guardar un objeto en una base de datos? Evidentemente hay que guardar por separado la parte de datos de cada objeto. Lógicamente, el código que implementa los métodos de las clases debe guardarse en la base de datos como parte de su esquema, junto con las definiciones de tipos de las clases. Sin embargo, muchas implementaciones sólo guardan el código en archivos ajenos a la base de datos para evitar tener que integrar el software del sistema, como los compiladores, con el sistema de bases de datos.

Hay varias maneras de hallar los objetos de la base de datos. Uno de los enfoques consiste en dar nombres a los objetos, igual que se hace con los archivos. Este enfoque resulta útil con un número de objetos relativamente pequeño, pero no resulta práctico para millones de objetos. Un segundo enfoque consiste en exponer los identificadores de los objetos o los punteros persistentes de los objetos, que pueden guardarse de manera externa. A diferencia de los nombres, estos punteros no tienen por qué ser fáciles de recordar y pueden ser incluso punteros físicos dentro de la base de datos.

Un tercer enfoque es guardar las colecciones de objetos y permitir que los programas iteren sobre las mismas para hallar los objetos deseados. Las colecciones de objetos pueden a su vez modelarse como objetos de un *tipo colección*. Los tipos de colecciones incluyen los conjuntos, los multiconjuntos (es decir, conjuntos con varias posibles apariciones de un mismo elemento), listas, etcétera. Un caso especial de colección son las *extensiones de clases*, que son la colección de todos los objetos pertenecientes a una clase. Si hay una extensión de clase para una clase dada, siempre que se cree un objeto de la clase, el mismo se inserta en la extensión de clase de manera automática, y siempre que se borre un objeto, éste se eliminará de la extensión de clase. Las extensiones de clases permiten que las clases se traten como relaciones en las que resulta posible examinar todos los objetos de una clase, igual que se pueden examinar todas las tuplas de una relación.

La mayor parte de los sistemas de bases de datos orientados a objetos permiten las tres maneras de acceso a los objetos persistentes. Todos los objetos tienen identificadores. Generalmente sólo se da nombre a las extensiones de las clases y a otros objetos de tipo colección y, quizás, a otros objetos seleccionados, pero no se da nombre a la mayor parte de los objetos. Las extensiones de las clases suelen conservarse para todas las clases que puedan tener objetos persistentes, pero, en muchas de las implementaciones, sólo contienen los objetos persistentes de cada clase.

8.5. SISTEMAS C++ PERSISTENTES

En los últimos años han aparecido varias bases de datos orientadas a objetos basadas en las extensiones de C++ persistentes (véanse las notas bibliográficas). Hay diferencias entre ellas en términos de la arquitectura de los sistemas pero tienen muchas características comunes en términos del lenguaje de programación.

Varias de las características orientadas a objetos del lenguaje C++ permiten proporcionar un buen soporte para la persistencia sin modificar el propio lenguaje. Por ejemplo, se puede declarar una clase denominada Persistent_Object (objeto persistente) con los atributos y los métodos para permitir la persistencia; cualquier otra clase que deba ser persistente puede hacerse subclase de esta clase y heredará, por tanto, permitir la persistencia. El lenguaje C++ (igual que otros lenguajes modernos de programación) permite también la posibilidad de redefinir los nombres y los operadores de las funciones estándar —como +, –, el operador de desreferencia de punteros, etcétera— en función del tipo de operandos a los que se aplica. Esta posibilidad se denomina *sobrecarga* y se utiliza para redefinir los operadores para que se comporten de la manera deseada cuando operan con objetos persistentes. En el siguiente apartado se ofrecen ejemplos del uso de estas características.

Proporcionar apoyo a la persistencia mediante las bibliotecas de clases presenta la ventaja de realizar cambios mínimos en C++ y de resultar relativamente fácil de implementar. Sin embargo, presenta el inconveniente de que los programadores tienen que emplear mucho más tiempo para escribir los programas que trabajen con objetos persistentes y de que no resulta sencillo especificar las restricciones de integridad del esquema ni permitir las consultas declarativas. Por tanto, la mayor parte de las implementaciones persistentes de C++ extienden la sintaxis de C++, por lo menos en cierto grado.

8.5.1. El lenguaje para la definición de objetos C++ de ODMG

El grupo de gestión de bases de datos de objetos (*Object Database Management Group*, ODMG) ha trabajado en la normalización de las extensiones de los lenguajes para que C++ y Smalltalk permitan la persistencia y en la definición de bibliotecas de clases con el mismo objetivo. ODMG publicó la primera versión de su norma en 1993. En 1997 el grupo publicó la segunda versión, ODMG-2.0, que se aborda en este capítulo.

La norma ODMG proporciona toda la funcionalidad mediante las bibliotecas de clases, sin ninguna extensión del lenguaje. Se describe la norma ODMG utilizando ejemplos. Hay que tener un conocimiento previo del lenguaje de programación C++ para comprender completamente los ejemplos. (Véanse las notas bibliográficas para hallar las referencias a los textos sobre C++.)

La extensión ODMG de C++ tiene dos partes: 1) el *lenguaje de definición de objetos* de C++ (C++ ODL, *Object Definition Language*) y 2) el *lenguaje de manipulación de objetos* de C++ (C++ OML, *Object Manipulation Language*). C++ ODL extiende la sintaxis de definición de tipos de C++.

En la Figura 8.7 se muestra un ejemplo de código escrito en C++ ODL de ODMG. Los esquemas de las cuatro clases se definen en el código. Cada clase se define como subclase de d_Object y, por tanto, los objetos de cada clase pueden hacerse persistentes. Las clases Sucursal, Cuenta y Persona son subclases directas de d_Object. La clase Cliente es una subclase de Persona y es también, por tanto, una subclase de d_Object. Los tipos d_String, d_Date, y d_Long están entre los tipos estándar definidos por ODMG-2.0. El tipo d_Long indica un entero de 32 bits.

C++ no permite de manera directa el concepto de mensajes. Sin embargo, los métodos se pueden llamar directamente y pueden comportarse como las llamadas a los procedimientos. La declaración de la clase cuenta ilustra los métodos calcular_saldo() y actualizar_saldo(d_Long delta), que se utilizan para leer y modificar el atributo saldo (no se ha presentado el código para

```
class Sucursal: public d_Object {
public:
    d_String nombre;
    d_String dirección;
    d_Long activo;
};

class Cuenta: public d_Object {
private:
    d_Long saldo;
public:
    d_Long número_cuenta;
    d_Set<d_Ref<Cliente>> titulares;
    d_Long calcular_saldo();
    int actualizar_saldo(d_Long delta);
};

class Persona: public d_Object {
public:
    d_String nombre;
    d_String dirección;
};

class Cliente: public Persona {
public:
    d_Date alta;
    d_Long id_cliente;
    d_Ref<Sucursal> sucursal_raíz;
    d_Set<d_Ref<Cuenta>> cuentas;
};
```

FIGURA 8.7. Ejemplo del lenguaje para la definición de objetos C++ de ODMG.

estos métodos). La declaración de la clase Cuenta muestra también las características de «encapsulamiento» de C++ (no específicas de ODMG). La palabra clave private indica que los atributos o los métodos siguientes sólo resultan visibles para los métodos de esa clase; public indica que los atributos o los métodos también son visibles para el resto del código. El atributo saldo de Cuenta se declara privado, lo que significa que ninguna función distinta de los métodos de esa clase puede leerlo o escribirlo. La clase Cuenta tiene otras dos variables que aparecen después de la palabra clave public, queriendo decir que pueden ser accedidas por otros códigos distintos de los métodos se la clase Cuenta.

El uso de la palabra clave public justo antes del nombre de una superclase indica que los atributos o los métodos públicos heredados de esa superclase siguen siendo públicos en la subclase. Estas características forman parte de C++ estándar, y no son específicas para ODMG.

El tipo d_Ref<Sucursal> utilizado en la clase Cliente es una *referencia*, o puntero persistente, a un objeto del tipo Sucursal. El tipo d_Set<d_Ref<Cliente>> utilizado en Cuenta es un conjunto de punteros persistentes a objetos del tipo Cuenta.

Las clases d_Ref<> y d_Set<> son clases *plantilla* definidas en la norma ODMG-2.0; las normas se definen de manera independiente de los tipos, y los programadores pueden crear ejemplares para los tipos necesarios (como d_Ref<Cuenta>) de la manera deseada.

Las referencias entre Cliente y Cuenta representan una relación entre los objetos Cliente y Cuenta. La relación se especifica en la siguiente restricción de integridad referencial: «Si un objeto Cliente tiene una referencia a un objeto Cuenta desde su atributo cuentas, debe haber también una **referencia inversa** desde el objeto Cuenta hasta el objeto Cliente a través del atributo titulares».

El programador puede especificar las restricciones de integridad referencial en ODMG-2.0, y el sistema automáticamente las mantendrá, de la forma en que se describe en los siguientes párrafos. El programador especifica la restricción de integridad referencial entre las clases Cuenta y Cliente declarando los atributos titulares y cuentas de las dos clases, como en la Figura 8.8. La norma define estos atributos como ejemplares de una clase plantilla d_Rel_Set<T,A>. Un ejemplar de esta clase contiene un conjunto de referencias a objetos de tipo T. Debe haber un atributo en la clase T que guarda la referencia inversa; el nombre de este atributo está guardado en A. Por restricciones del lenguaje C++, no se puede usar directamente la cadena «titulares» o «cuentas» como un parámetro de la clase plantilla, así que se se usan las variables titulares y cuentas, que guardan las cadenas «titulares» y «cuentas» respectivamente.

ODMG-2.0 conserva la integridad referencial automáticamente: la clase plantilla d_Rel_Ref<T,A> proporciona un método para la inserción de una referencia; este método también inserta una referencia inversa (si

```
extern const char _titulares[], _cuentas[];
class Cliente: public Persona {
public:
    …
    d_Rel_Set<Cuenta,_titulares> cuentas;
};

class Cuenta: public d_Object {
public:
    …
    d_Rel_Set<Cliente, _cuentas> titulares;
    …
};
const char _titulares[] = «titulares»;
const char _cuentas[] = «cuentas»;
```

FIGURA 8.8. Restricciones de integridad referencial en ODMG C++ ODL.

no está ya presente) en el atributo, cuyo nombre se guarda en A, de la clase T. Por ejemplo, para añadir una nueva cuenta a un cliente se inserta una referencia a la cuenta en el atributo cuentas de Cliente; usando el método para insertar de la clase d_Rel_Set<Cuenta,_titulares>. Este método conserva la restricción de integridad referencial insertando automáticamente una referencia a titular en la variable titulares de la cuenta. Análogamente, cuando el sistema lleva a cabo un borrado, el método correspondiente borra la referencia inversa. De este modo, el método garantiza que la restricción de integridad referencial no se viola.

ODMG-2.0 también proporciona una clase plantilla d_Rel_Ref<T,A> que proporciona la integridad referencial a los atributos que guardan una única referencia. En este caso, la asignación a la variable de tipo d_Rel_Ref<T,A> se redefine para conservar automáticamente la referencia inversa. Supóngase que el objeto Sucursal tiene un atributo clientes que guardan la referencia inversa al objeto Cliente. A continuación se puede declarar el atributo sucursal_raíz de Cliente que será del tipo d_Rel_Ref<Sucursal,_clientes>, donde _clientes es una variable inicializada a la cadena «clientes». El atributo clientes de Sucursal sería del tipo d_Rel_Set<Cliente,_sucursal_raíz>, donde sucursal_raíz es una variable inicializada a la cadena «sucursal_local». Así, las restricciones de integridad referencial se conservan automáticamente.

8.5.2. El lenguaje para la manipulación de objetos C++ de ODMG

En la Figura 8.9 se muestra código escrito en C++ OML de ODMG. Primero, el programa abre una base de datos y a continuación se inicia una transacción. Recuérdese el concepto de transacción del Apartado 4.9.5; las transacciones forman una unidad de trabajo que es atómica; es decir, si la transacción no se completa con éxito, el sistema deshace el efecto de la transacción.

Como paso siguiente, el programa crea un objeto cuenta y un objeto titular usando el operador new. La

```
int crear_titular_cuenta(String nombre, String dirección) {
    d_Database bd_banco_obj;
    d_Database * bd_banco = &bd_banco_obj;
    bd_banco->open(«BD-Banco»);
    d_Transaction Trans;
    Trans.begin ();

    d_Ref<Cuenta> cuenta = new(bd_banco,«Cuenta»)Cuenta;
    d_Ref<Cliente> clien = new(bd_banco,«Cliente») Cliente;
    clien->nombre = nombre;
    clien->dirección = dirección;
    clien->cuentas.insert_element(cuenta);
    cuenta->titulares.insert_element(clien);
    .. Código para inicializar id_cliente, número de cuenta, etc.
    Trans.commit();
    bd_banco->close();
}
```

FIGURA 8.9. Ejemplo del lenguaje para la manipulación de objetos C++ de ODMG.

clase d_Object implementa varios métodos, incluyendo la versión persistente del operador de asignación de memoria new que se utiliza en el código de ejemplo. Esta versión del operador new asigna el objeto a la base de datos especificada en vez de en la memoria. El operador también toma un parámetro que especifica el nombre de la clase del objeto que está siendo asignado; el nombre de la clase se usa para seguir la trayectoria de qué objetos pertenecen a una determinada clase en una base de datos.

El programa entonces inicializa los objetos cuenta y titular. Se utiliza el método insert_element de la clase plantilla d_Set<> para insertar las referencias a los clientes y a las cuentas en los conjuntos adecuados después de crear los objetos cliente y cuenta. Si se ha declarado cuenta y titular de tipo d_Rel_Set, tan pronto como una cuenta se añada al conjunto de cuentas de un cliente, las referencias inversas desde el objeto cuenta (a través del atributo titular) se crearán automáticamente. De este modo, la inserción del cliente en el conjunto de titulares de la cuenta se vuelve innecesario (aunque no incorrecto). De manera parecida, si se elimina una cuenta de un cliente, el conjunto de titulares de la cuenta se actualizaría automáticamente, borrando al cliente.

Al final, el programa compromete la transacción y cierra la base de datos. Una **transacción** es una secuencia de pasos, delimitados por una llamada a begin (iniciar la transacción) y otra llamada a commit (comprometer) o abort (abortar). Los pasos de la transacción forman una unidad atómica. Es decir, el sistema de bases de datos garantiza que 1) si se ejecuta la operación commit() para una transacción, todas las actualizaciones hechas persistirán en la base de datos, y 2) si se ejecuta una operación abort(), o si el programa que está ejecutando la transacción termina sin ejecutar commit(), todas las actualizaciones representadas como parte de la transacción serán deshechas. Si hay algún fallo antes de que una transacción se comprometa, el sistema deshace todas las modificaciones de la transacción hasta el último estado estable.

8.5.2.1. Extensiones de clases

En el código de la Figura 8.9, los identificadores de los objetos cliente y cuenta no están explícitamente guardados en ningún sitio, aunque los objetos están no obstante en la base de datos. La cuestión natural es, una vez que el programa que creó estos objetos termina, ¿cómo acceder a los objetos? Hay dos formas de hacer esto.

En el primer método, la norma ODMG proporciona un mecanismo para especificar, como parte del esquema de una clase en una base de datos, que una extensión de clase debe mantenerse para la clase. Esta especificación se puede hacer mediante herramientas administrativas proporcionadas por una implementación ODMG o por una llamada a una función. Una vez se hace esta especificación, si un programador crea un objeto de la clase, digamos C, en esta base de datos, el sistema añade el identificador del objeto a la extensión de clase de la clase C. Si un programador borra un identificador de objeto, el sistema borra sus identificadores desde la extensión de clase de C. La clase plantilla d_Extent se usa para acceder a los elementos en una extensión de clase. En el Apartado 8.5.2.2 se resume la forma de hacerlo.

El segundo método es dar nombres a los objetos, igual que se dan nombres a los ficheros en un sistema de ficheros. Puesto que una base de datos puede contener un gran número de objetos de cada clase, dar nombres a todos los objetos se vuelve problemático. En su lugar, se puede crear un conjunto persistente que contenga los identificadores de todos los objetos en una clase, y dar un nombre al conjunto. En otras palabras, se crea manualmente una extensión de clase para la clase. Se describe con detalle en el Apartado 8.5.2.4.

8.5.2.2. Iteradores

Se puede iterar en una colección de referencias utilizando un *iterador*. Se crea un iterador con el método create_iterador() proporcionado por la clase colección, como d_Set, igual que por la clase especial d_Extent, que se utiliza para acceder a la extensión de clase.

En el código de la Figura 8.10, la línea

```
d_Extent<Cliente>todos_los_clientes(bd_banco);
```

declara todos_los_clientes para ser una extensión de clase de la clase Cliente, y lo inicializa para ser la extensión de clase de Clientes en la base de datos bd_banco. A continuación la variable iter se establece para ser un iterador sobre los objetos en la extensión de clase de Cliente.

El método next(), proporcionado por el iterador, se utiliza para avanzar por los elementos consecutivos de la colección de clientes. Para cada cliente se llama al método mostrar_clien (que se supone que se ha definido en alguna otra parte) para mostrar el cliente.

```
int mostrar_clientes (){
    d_Database bd_banco_obj;
    d_Database * bd_banco = &bd_banco_obj;
    bd_banco->open(«BD-Banco»);
    d_Transaction Trans;
    Trans.begin();
    d_Extent<Cliente>todos_los_clientes(bd_banco);

    d_Iterator<d_Ref<Cliente>>iter=todos_los_clientes.create_iterator();
    d_Ref<Cliente>p;
    while(iter.next(p)) {
        mostrar_clien(p);
    }
    Trans.commit();
}
```

FIGURA 8.10. Ejemplo de utilización de los iteradores de C++ de ODMG.

Las clases colección y la clase d_Extent proporcionan también un método select(), que es parecido a create_iterador(), pero toma una condición de selección como argumento, e itera solamente sobre aquellos objetos que satisfacen la condición de selección.

8.5.2.3. Modificación de objetos

Si un objeto va a ser modificado, la norma ODMG requiere que el sistema de bases de datos sea notificado del cambio. Para hacer esto, el programa debe invocar al método mark_modified() sobre el objeto antes de que sea modificado.

Por ejemplo, el método actualizar_saldo(d_Long delta) de la clase Cuenta debe invocar a mark_modified() antes de la actualización de su saldo. Si no se hace así puede que la actualización no se refleje en la base de datos.

Hasta ahora, nuestro programa ejemplo sólo ha modificado objetos que justo han sido creados, y no hay necesidad de invocar a mark_modified() para tales objetos. Los métodos definidos por el sistema como insert_element() invocan automáticamente a mark_modified().

La necesidad de marcar los objetos como modificados antes de modificarlos puede conducir a errores, puesto que los programadores pueden olvidarse fácilmente de hacerlo. Algunas bases de datos orientadas a objetos implementan técnicas para determinar automáticamente cuándo se ha modificado un objeto, liberando al programador de esta tarea.

8.5.2.4. Creación manual de extensión de clases

Como ejercicio de escritura de un programa en la norma ODMG, considérese cómo crear manualmente una extensión de clase para el objeto Cliente. Es conveniente guardar el identificador del conjunto en una variable global asociada con la clase Cliente. Se declara la variable global añadiendo la siguiente línea como parte de la declaración de la clase Cliente.

```
static d_Ref<d_Set<d_Ref<Cliente>>>
        todos_los_clientes;
```

Se asigna inicialmente el conjunto persistente de Cliente en la base de datos y guarda su identificador en la variable global como sigue:

```
Cliente::todos_los_clientes =
        new(bd_banco) d_Set<d_Ref<Cliente>>;
```

Cuando se crea un objeto Cliente, se inserta su identificador en el conjunto de clientes utilizando una instrucción de la forma

```
Cliente::todos_los_clientes->insert_element(clien);
```

Se puede crear un conjunto persistente para guardar los identificadores de los objetos Cuenta y guardar los identificadores de los objetos cuenta en él de manera similar. Sin embargo, el valor de la variable Cliente::todos_los_clientes, y por tanto el identificador del conjunto persistente, desaparecería cuando finalizara la transacción. Para encontrar el conjunto en la base de datos más tarde, cuando se asigna inicialmente el conjunto se le da un nombre al identificador de objeto en la base de datos como sigue:

```
bd_banco->set_object_nombre
    (Cliente::todos_los_clientes,«Todos_los_clientes»);
```

Cuando arranca una transacción posterior, primero abre la base de datos y a continuación busca el conjunto creado antes e inicializa Cliente::todos_los_clientes

```
Cliente::todos_los_clientes =
      bd_banco->lookup_object(«todos_los_clientes»);
```

Una **constructora** de una clase es un método especial que se utiliza para inicializar los objetos cuando se crean; se llama de manera automática cuando se ejecuta el operador new. De manera parecida, un **destructor** de una clase es un método especial que se llama cuando se borran los objetos de esa clase. Añadiendo la instrucción insert_element a las constructoras de una clase y la correspondiente instrucción delete_element a los des-

tructores de la clase, un programador puede asegurar que la colección Cliente::todos_los_clientes se mantendrá de la manera adecuada. Como se observó en el Apartado 8.5.2.1, las implementaciones de la norma ODMG pueden mantener las extensiones de clases automáticamente, y de este modo no es necesario mantenerlas manualmente.

8.5.2.5. *Lenguaje de consulta de objetos*

La norma ODMG proporciona el lenguaje de consultas declarativo OQL. OQL presenta el aspecto de SQL. El siguiente código crea y ejecuta una cuestión OQL para encontrar todas las cuentas pertenecientes a «Santos» cuyo saldo es mayor que 100. El resultado se guarda en la variable de tipo conjunto resultado.

```
d_Set<d_Ref<Cuenta>> resultado;
d_OQL_Query q1(«select a
                from Cliente c, c.cuentas a
                where c.nombre = 'Santos'
                   and a.obtener_saldo() > 100»);
d_oql_execute(q1,resultado);
```

8.5.2.6. *Cómo hacer transparente la persistencia de punteros*

Un inconveniente en el enfoque ODMG es que el programador tiene que tratar con dos tipos de punteros diferentes —punteros en memoria y referencias d_Ref— y el código que se escribe para un tipo no será válido para trabajar con el otro. Además, el programador tiene obligatoriamente que hacer la llamada mark_modified() cuando se modifique un objeto.

En cambio, el sistema de bases de datos **ObjectStore** permite la referencia a los objetos de la base de datos utilizando el estilo de los punteros habituales en C++, en vez de utilizar d_Ref. Para el programador, los punteros a objetos persistentes son como los punteros a objetos en memoria habituales. La persistencia de punteros es transparente, y sólo tienen un tipo puntero que hacen la programación mucho más fácil. Los programadores de ObjectStore escriben los programas exactamente como con el lenguaje C++ habitual con la única diferencia de que, como en ODMG, los objetos pueden crearse en la base de datos utilizando una forma especial de new.

Esta funcionalidad se implementa a través de un mecanismo de traslación llamado *rescate*, que hace la conversión entre punteros persistentes y punteros a memoria; el *rescate* se describe en el Capítulo 11.

El sistema de bases de datos ObjectStore también proporciona extensiones del lenguaje C++ que permite especificar más fácilmente las restricciones de integridad referencial. Los cambios también se detectan automáticamente y no requiere las llamadas a mark_modified().

8.6. SISTEMAS JAVA PERSISTENTES

En años recientes el lenguaje Java ha visto un enorme crecimiento en su uso. La demanda para permitir la persistencia de datos en los programas Java se ha incrementado correspondientemente, y el consorcio ODMG ha definido las normas para que se permita la persistencia en Java.

El modelo ODMG para la persistencia de objetos en programas Java es diferente del modelo para la persistencia que se permite en programas C++. La mayor diferencia es el uso de la **persistencia por alcance** en Java. Los objetos no se crean explícitamente en la base de datos. En su lugar, se dan los nombres a los objetos en la base de datos que sirven como **raíces** para la persistencia. Estos objetos, y algunos objetos alcanzables desde estos objetos, son persistentes.

La persistencia por alcance implica que los punteros persistentes deben ser del mismo tipo que los punteros transitorios, por lo que no hay equivalencia de la clase plantilla de C++ d_Ref. Otro resultado de la persistencia por alcance es que los objetos en la base de datos pueden volverse basura si el objeto se vuelve inalcanzable desde todas las raíces persistentes en la base de datos, y ninguna transacción activa guarda un puntero hacia el objeto. Tales objetos deben ser eliminados ejecutando periódicamente un procedimiento de **recogida de basura** en la base de datos. La recogida de basura se efectúa generalmente de forma concurrente con otras actividades de la base de datos.

Si un objeto de una clase se alcanza desde una raíz persistente, la clase se debe hacer *persistente*. Una clase normalmente se hace persistente (es decir, los objetos de esta clase pueden volverse persistentes) ejecutando un postprocesador en el código de la clase generado por la compilación del programa Java. También es posible hacer persistente manualmente una clase insertando instrucciones apropiadas en el código Java, pero este enfoque es relativamente complicado y no se recomienda.

El postprocesador también inserta código para marcar automáticamente los objetos como modificados si son actualizados, a diferencia de la vinculación C++ de la norma ODMG en la que los objetos que se modifican se deben marcar explícitamente usando mark_modified().

Cuando una transacción desea acceder a los objetos en la base de datos, se debe comenzar por uno de los objetos raíz de la base de datos que la transacción busca por su nombre. El sistema va por el objeto raíz des-

de la base de datos en memoria. Las implementaciones pueden elegir ir a buscar todos los objetos referenciados inmediatamente desde el objeto raíz, o buscar los objetos referenciados de una forma perezosa. En una búsqueda perezosa de objetos, el objeto raíz es inicialmente «vacío», es decir, se asigna una localización de memoria pero sus campos no se inicializan. Cuando se accede por primera vez un objeto vacío *h*, sus campos se rellenan desde la base de datos. El objeto vacío *h* puede contener referencias a otros objetos; si cualquiera de estos objetos se encuentra ya en memoria, sus direcciones en memoria se usan para reemplazar la referencia en *h*. Si el objeto referenciado no está en memoria, se crea para un objeto vacío ello, y la dirección en memoria del objeto vacío se usa para reemplazar la referencia en *h*.

La norma ODMG para Java define una colección de tipos tales como DSet, DBag y DList que extienden la colección de tipos estándar en Java. Java ya define un tipo iterador para iterar sobre colecciones.

8.7. RESUMEN

- Las aplicaciones de bases de datos de la generación actual no encajan a menudo en el conjunto de suposiciones hecho para el tipo de aplicaciones más antiguas de procesamiento de datos. Se ha desarrollado el modelo de datos orientado a objetos para trabajar con varios de estos nuevos tipos de aplicaciones.
- El modelo de datos orientado a objetos es una adaptación para los sistemas de bases de datos del paradigma de la programación orientada a objetos. Se basa en el concepto de encapsular los datos en un objeto y el código que opera sobre ellos.
- De manera parecida, los objetos estructurados se agrupan en clases. El conjunto de las clases se estructura en subclases y superclases basadas en una extensión del concepto ES del modelo entidad-relación.
- El valor de un elemento de datos de un objeto puede ser un objeto, haciendo posible representar los continentes de objetos, lo que da lugar a objetos compuestos.
- Hay dos enfoques para la creación de bases de datos orientadas a objetos: se pueden añadir los conceptos de la programación orientada a objetos a los lenguajes de bases de datos existentes o extender los lenguajes orientados a objetos existentes para que trabajen con las bases de datos añadiendo conceptos como la persistencia y las colecciones. Las bases de datos relacionales extendidas adoptan el primer enfoque. Los lenguajes de programación persistentes adoptan el segundo.
- Las extensiones persistentes de C++ y Java han logrado progresos significativos en los últimos años. La integración de la persistencia sin solución de continuidad y de manera ortogonal con las constructoras de lenguajes existentes es importante para su facilidad de uso.
- La norma ODMG define clases y otras constructoras para la creación y acceso a los objetos persistentes desde C++ y Java.

TÉRMINOS DE REPASO

- Ambigüedad en la herencia
- C++ ODL de ODMG
  — d_Ref, d_Set
  — Integridad referencial
    – _Ref_Sel, d_Ref_Ref
- C++ ODL de ODMG
  — Extensión de Clase (d_Extent)
  — Iteradores (d_iterator)
  — Lenguaje de consulta de objetos
- Clase
- Clase más específica
- Continente de objetos
- Ejemplar
- Encapsulamiento
- Grafo acíclico dirigido
- Herencia
- Herencia múltiple
- Identidad de objeto
  — Incorporada
  — Nombre
  — Valor
- Identificador de objeto
- Jerarquía de clases
- Lenguajes de programación persistente
- Mensajes
- Métodos
- ObjectStore

- Objeto
- ODMG Java
  - Persistencia por alcance
  - Raíces
  - Recogida de basura
- Papeles
- Persistencia por
  - Alcance
  - Clase
  - Creación
  - Marca
- Posibilidad de sustitución
- Subclase
- Superclase
- Variables

EJERCICIOS

8.1. Para cada una de las siguientes áreas de aplicación explíquese la razón por la que no resultan apropiados los sistemas de bases de datos relacionales. Indíquense todos los componentes específicos del sistema que habría que modificar.

a. Diseño asistido por computadora.
b. Bases de datos multimedia.

8.2. ¿En qué se diferencian el concepto de objeto del modelo orientado a objetos y el concepto de entidad del modelo entidad-relación?

8.3. Una compañía de alquiler de coches tiene una base de datos de los vehículos de su flota actual. Para todos los vehículos incluye el número de identificación de cada uno, el número de la matrícula, el fabricante, el modelo, la fecha de adquisición y el color. Se incluyen datos específicos para algunos tipos de vehículos:

- Camiones: capacidad de carga.
- Coches deportivos: potencia, requisitos de edad del conductor.
- Camionetas: número de plazas.
- Vehículos todo terreno: altura de los bajos, eje motor (tracción a dos o a las cuatro ruedas).

Constrúyase la definición del esquema de una base de datos orientada a objetos para esta base de datos. Utilícese la herencia donde resulte conveniente

8.4. Explíquese el motivo de que pueda haber ambigüedad en caso de herencia múltiple. Ilústrese la explicación con un ejemplo.

8.5. Explíquese la diferencia entre el concepto de identidad de los objetos del modelo orientado a objetos y el concepto de igualdad de las tuplas del modelo relacional.

8.6. Explíquese la diferencia de significado entre los arcos de un grafo dirigido acíclico que represente la herencia y un grafo dirigido acíclico que represente los continentes de objetos.

8.7. ¿Por qué permiten los lenguajes de programación persistentes los objetos transitorios? ¿Sería más sencillo utilizar sólo objetos persistentes y borrar los objetos no necesarios al concluir la ejecución? Explíquese la respuesta.

8.8. Utilizando C++ de ODMG

a. Dense definiciones de esquemas correspondientes al esquema relacional que se muestra en la Figura 3.39 usando referencias para expresar las relaciones de clave externa.
b. Escríbanse programas para resolver cada una de las cuestiones del ejercicio 3.10.

8.9. Utilizando C++ de ODMG, dense definiciones de esquema correspondientes al diagrama E-R de la Figura 2.29. Utilícense referencias para implementar las relaciones.

8.10. Explíquese, utilizando un ejemplo, cómo representar una relación ternaria en un modelo de datos orientado a objetos tal como C++ de ODMG.

8.11. Explíquese la manera en que se implementa un puntero persistente. Compárese esta implementación con la de los punteros de los lenguajes de propósito general, como C o Pascal.

8.12. Si se crea un objeto sin que haya referencias al mismo, ¿cómo se puede borrar?

8.13. Considérese un sistema que proporcione objetos persistentes. ¿Se trata necesariamente de un sistema de bases de datos? Explíquese la respuesta.

NOTAS BIBLIOGRÁFICAS

Las aplicaciones de los conceptos de las bases de datos a CAD se discuten en Haskin y Lorie [1982] y Lorie et al.[1985].

La programación orientada a objetos se discute en Goldberg y Robson [1983], Stefik y Bobrow [1986] y Stroustrup [1988]. Stroustrup [1997] describe el lenguaje de programación C++.

Hay numerosos sistemas de bases de datos orientados a objetos ya implementados, incluyendo (por orden alfabético) Cactis (Hudson y King [1989]); E/Exodus,

desarrollado en la Universidad de Wisconsin (Carey et al. [1990]); Gemstone (Maier et al. [1986]); Iris, desarrollado en Hewlett-Packard (Fishman et al. [1990]); Jasmine, desarrollado en los Laboratorios Fujitsu (Ishikawa et al. [1993]); O_2 (Lecluse et al. [1988]); ObjectStore (Lamb et al. [1991]); Ode, desarrollado en los Laboratorios Bell (Agrawal y Gehani [1989]); Ontos; Open-OODB; Orion (Kim et al. [1988]); Versant y otros.

La norma de bases de datos orientadas a objetos ODMG se describe con detalle en Cattell [2000].

En Kim [1990], Zdonik y Maier [1990] y Dogac et al. [1994] se pueden encontrar visiones de conjunto de la investigación en bases de datos orientadas a objetos. La identidad de objetos se caracteriza en detalle por Khoshafian y Copeland [1990].

La modificación de los esquemas en las bases de datos orientadas a objetos es más complicada que en las bases de datos relacionales, dado que las bases de datos orientadas a objetos tienen sistemas de tipos complejos con herencia. La modificación de los esquemas se discute en Skarra y Zdonik [1986], Banerjee et al. [1987] y en Penney y Stein [1987].

Goodman [1995] describe las ventajas y los inconvenientes de utilizar bases de datos orientadas a objetos en una aplicación para la base de datos del genoma. Lunt [1995] proporciona una visión en conjunto de los aspectos de la autorización en las bases de datos orientadas a objetos.

CAPÍTULO

9 BASES DE DATOS RELACIONALES ORIENTADAS A OBJETOS

Los lenguajes de programación persistentes añaden la persistencia y otras características de las bases de datos a los lenguajes de programación existentes con sistemas de tipos orientados a objetos. Por el contrario, los *modelos de datos relacionales orientados a objetos* extienden el modelo de datos relacional proporcionando un sistema de tipos más rico e incluyendo tipos de datos complejos y la programación orientada a objetos. Los lenguajes de consulta relacionales como SQL también necesitan ser extendidos para trabajar con el sistema de tipos enriquecido. Estas extensiones intentan conservar los fundamentos relacionales —en concreto, el acceso declarativo a los datos— al tiempo que extienden la capacidad de modelado. Los sistemas de bases de datos relacionales orientados a objetos (es decir, los sistemas de bases de datos basados en el modelo objeto-relación) proporcionan un modo de cambio adecuado para los usuarios de las bases de datos relacionales que deseen utilizar características orientadas a objetos.

En primer lugar, se presenta la motivación del modelo relacional anidado, que permite relaciones que no cumplen la primera forma normal y permite la representación directa de las estructuras jerárquicas. Posteriormente se muestra la manera de extender SQL añadiendo varias características relacionales orientadas a objetos. El estudio se basa en la norma SQL:1999.

Finalmente se analizan las diferencias entre los lenguajes de programación persistentes y los sistemas relacionales orientados a objetos y se mencionan los criterios para la elección entre unos y otros.

9.1. RELACIONES ANIDADAS

En el Capítulo 7 se definió la *primera forma normal* (1FN), que exige que todos los atributos tengan *dominios atómicos*. Un dominio es *atómico* si los elementos del mismo se consideran unidades indivisibles.

La suposición de 1FN es natural en el ejemplo bancario considerado en capítulos anteriores. Sin embargo, no todas las aplicaciones se modelan de la mejor forma con relaciones 1FN. Por ejemplo, en lugar de ver la base de datos como un conjunto de registros, los usuarios de ciertas aplicaciones deben tratarla como un conjunto de objetos (o entidades). Estos objetos pueden requerir una correspondencia uno a uno entre la noción intuitiva del usuario de un objeto y la noción del sistema de bases de datos de un elemento de datos.

El **modelo relacional anidado** es una extensión del modelo relacional en la que los dominios pueden ser atómicos o de relación. Por tanto, el valor de las tuplas de los atributos puede ser una relación, y las relaciones pueden guardarse en otras relaciones. Los objetos complejos, por tanto, pueden representarse mediante una única tupla de las relaciones anidadas. Si se consideran las tuplas de las relaciones anidadas como elementos de datos, se tiene una correspondencia uno a uno entre los elementos de datos y los objetos de la vista de la base de datos del usuario.

Las relaciones anidadas se ilustran mediante un ejemplo extraído de una biblioteca. Considérese que para cada libro se almacena la información siguiente:

- Título del libro
- Lista de autores
- Editorial
- Lista de palabras clave

título	*lista-autores*	*editorial (nombre, sucursal)*	*lista-palabras-clave*
Compiladores	{Gómez, Santos}	(McGraw-Hill, Nueva York)	{traducción, análisis}
Redes	{Santos, Escudero}	(Oxford, Londres)	{Internet, Web}

FIGURA 9.1. La relación de documentos *libros* que no está en 1FN.

Puede verse que, si se define una relación para la información anterior, varios de los dominios serán no atómicos.

- **Autores.** Un libro puede tener varios autores. No obstante, puede que se desee hallar todos los documentos entre cuyos autores estuviera Santos. Por tanto, hay interés en una parte del elemento del dominio «conjunto de autores».
- **Palabras clave.** Si se guarda un conjunto de palabras clave de cada documento se espera poder recuperar todos los documentos cuyas claves incluyan una o varias de las palabras clave especificadas. Por tanto, se considera que el dominio de la lista de palabras clave no es atómico.
- **Editorial.** A diferencia de *palabras clave* y *autores*, *editorial* no tiene un dominio de tipo conjunto. Sin embargo, se puede considerar que *editorial* consiste en los subcampos *nombre* y *sucursal*. Esta manera de considerarlo hace que el dominio de *editorial* no sea atómico.

En la Figura 9.1 se muestra un ejemplo de la relación de documentos *libros*. La relación *libros* puede representarse en 1FN como se muestra en la Figura 9.2. Dado que en 1FN hay que disponer de dominios atómicos pero se desea tener acceso a los diferentes autores y palabras clave, hace falta una tupla para cada par (palabra clave, autor). El atributo *editorial* se sustituye en la versión 1FN por dos atributos: uno por cada subcampo de *editorial*.

Gran parte de la incomodidad de la relación *libros-planos* de la Figura 9.2 se elimina si se supone que se cumplen las dependencias multivaloradas siguientes:

título →→ *autor*
título →→ *palabra-clave*
título → *nombre-editorial*, *sucursal-editorial*

Por tanto, se puede descomponer la relación en 4FN utilizando los esquemas:

autores(*título*, *autor*)
palabras-clave(*título*, *palabra-clave*)
libros4(*título, nombre-editorial, sucursal-editorial*)

En la Figura 9.3 se muestra la proyección de la relación *libros-planos* de la Figura 9.2 en la descomposición anterior.

Aunque el ejemplo de la base de datos de libros se puede expresar adecuadamente sin usar relaciones anidadas, su uso conduce a un modelo más fácil de comprender, dado que el usuario típico de los sistemas de recuperación de información piensa en la base de datos en términos de libros que tienen una lista de autores, como los modelos de diseño que no están en 1FN. El diseño 4FN necesita que los usuarios incluyan reuniones en las consultas, lo que complica la interacción con el sistema.

Se puede definir una vista relacional no anidada (cuyo contenido sea indéntico a *libros-planos*) que elimine la necesidad de que los usuarios escriban reuniones en las consultas. En esa vista, sin embargo, se pierde la correspondencia uno a uno entre las tuplas y los documentos.

título	*autor*	*nombre-editorial*	*sucursal-editorial*	*palabra-clave*
Compiladores	Gómez	McGraw-Hill	Nueva York	traducción
Compiladores	Santos	McGraw-Hill	Nueva York	traducción
Compiladores	Gómez	McGraw-Hill	Nueva York	análisis
Compiladores	Santos	McGraw-Hill	Nueva York	análisis
Redes	Santos	Oxford	Londres	Internet
Redes	Escudero	Oxford	Londres	Internet
Redes	Santos	Oxford	Londres	Web
Redes	Escudero	Oxford	Londres	Web

FIGURA 9.2. *libros-planos*, una versión 1FN de la relación *libros* que no estaba en 1FN.

9.2. TIPOS COMPLEJOS

Las relaciones anidadas son sólo un ejemplo de las extensiones del modelo relacional básico. Otros tipos de datos no atómicos, como los registros anidados, también se han mostrado útiles. El modelo de datos orientado a objetos ha creado la necesidad de características como la herencia y las referencias a los objetos. Los sistemas de tipos complejos y la programación orientada a objetos permiten que los conceptos del modelo E-R, como la identidad de las entidades, los atributos multivalorados y la generalización y la especialización, se representen directamente sin que haga falta una compleja traducción al modelo relacional.

En este apartado se describen las extensiones de SQL para que permita los tipos complejos, incluyendo las relaciones anidadas y las características orientadas a objetos. La presentación se basa en la norma SQL:1999, pero también se describen características que no están actualmente en la norma pero que pueden ser introducidas en futuras versiones de la norma SQL.

título	*autor*
Compiladores	Gómez
Compiladores	Santos
Redes	Santos
Redes	Escudero

autores

título	*palabra-clave*
Compiladores	beneficios
Compiladores	estrategia
Redes	beneficios
Redes	personal

palabras-clave

título	*nombre-editorial*	*sucursal-editorial*
Compiladores	McGraw-Hill	Nueva York
Redes	Oxford	Londres

libros4

FIGURA 9.3. Versión 4FN de la relación *libros-planos* de la Figura 9.2.

9.2.1. Tipos colección y tipos de objetos de gran tamaño

Considérese este fragmento de código.

create table *libros* (
...
lista-palabras-clave **setof**(**varchar**(20))
...
)

Esta definición de tabla es diferente de las definiciones en las bases de datos relacionales normales, ya que permite que los atributos sean **conjuntos**, permitiendo que los atributos multivalorados de los diagramas E-R se representen directamente.

Los conjuntos son ejemplares de los **tipos colección**. Otros ejemplares son los **arrays** y los **multiconjuntos** (es decir, colecciones sin orden donde un elemento puede aparecer varias veces). Las siguientes definiciones de atributos ilustran la declaración de un *array*:

array-autores **varchar**(20) **array** [10]

array-autores es un *array* de hasta 10 nombres de autor. Se puede acceder a los elementos del *array* especificando el índice del *array*, por ejemplo, *array-autores*[1].

Los *arrays* son el único tipo colección soportado en SQL:1999; la sintaxis usada es como en la declaración precedente. SQL:1999 no da soporte a conjuntos sin orden o multiconjuntos, aunque es posible que aparezcan en versiones futuras de SQL[1].

Muchas aplicaciones actuales de bases de datos necesitan almacenar atributos grandes (del orden de varios kilobytes), tales como la fotografía de una persona, o muy grandes (del orden de varios megabytes o incluso gigabytes), tales como imágenes médicas de alta resolución o clips de vídeo. SQL:1999 proporciona por tanto nuevos tipos de datos para objetos de gran tamaño para datos de caracteres (**clob**) y binarios (**blob**). Las letras «lob» en estos tipos de datos son acrónimos de «Large OBject» (objeto grande). Por ejemplo, se pueden declarar los siguientes atributos:

crítica-libro **clob**(10KB)
imagen **blob**(10MB)
película **blob**(2GB)

Los objetos grandes se usan normalmente en aplicaciones externas, y tiene poco sentido extraerlos completamente en SQL. En su lugar, una aplicación conseguiría un «localizador» de un objeto grande y lo usaría para manipularlo desde el lenguaje anfitrión. Por ejemplo, JDBC permite al programador extraer un objeto grande en pequeños trozos, en lugar de todo a la vez, de forma muy parecida a la extracción de datos de un archivo del sistema operativo.

9.2.2. Tipos estructurados

Los tipos estructurados se pueden declarar y usar en SQL:1999 como en el siguiente ejemplo:

create type *Editorial* **as**
(*nombre* **varchar**(20),
sucursal **varchar**(20))
create type *Libro* **as**
(*título* **varchar**(20),
array-autores **varchar**(20) **array** [10],
fecha-pub **date**,
editorial Editorial,
lista-palabras-clave **setof**(**varchar**(20)))
create table *libros* **of type** *Libro*

La primera instrucción define el tipo *Editorial*, que tiene dos componentes: un nombre y una sucursal. La segunda instrucción define el tipo *Libro*, que contiene *título*, *array-autores*, que es un *array* de autores, una fecha de publicación, una editorial (de tipo *Editorial*) y un conjunto de palabras clave. (La declaración de *lista-palabras-clave* como un conjunto usa la sintaxis extendida y no está soportada en la norma SQL:1999.) Los tipos ilustrados se denomina **tipos estructurados** en SQL:1999.

Finalmente, se crea la tabla *libros* que contiene tuplas del tipo *Libro*. La tabla es similar a la relación anidada *libros* de la Figura 9.1, excepto en que se ha decidido crear un *array* de nombres de autores en lugar de un conjunto. El *array* permite registrar el orden de los nombres de autores.

Los tipos estructurados permiten la representación directa de atributos compuestos de los diagramas E-R. También se pueden usar **tipos fila** en SQL:1999 para

[1] El sistema de bases de datos Oracle 8 soporta relaciones anidadas, pero usa una sintaxis diferente de la de este capítulo.

definir atributos compuestos. Por ejemplo, se podría haber definido un atributo *editorial1* como

```
editorial1 row (nombre varchar(20),
                sucursal varchar(29))
```

en lugar de crear un tipo con nombre *Editorial*.

Por supuesto se pueden crear tablas sin crear un tipo intermedio para la tabla. Por ejemplo, la tabla *libros* se podría también definir como:

```
create table libros
    (título varchar(20),
    array-autores varchar(20) array [10],
    fecha-pub date,
    editorial Editorial,
    lista-palabras-clave setof(varchar(20)))
```

Con esta declaración no hay un tipo explícito para las filas de la tabla[2].

Un tipo estructurado puede tener **métodos** definidos sobre él. Los métodos se declaran como parte de la definición de tipos de un tipo estructurado.

```
create type Empleado as (
    nombre varchar(20),
    sueldo integer)
method incrementar(porcentaje integer)
```

El cuerpo del método se crea separadamente:

```
create method incrementar(porcentaje integer)
        for Empleado
begin
    set selft.sueldo = self.sueldo + (self.sueldo*
                porcentaje)/100
end
```

La variable **self** se refiere al ejemplar del tipo estructurado sobre el que se invoca el método. El cuerpo del método puede contener instrucciones procedimentales, que se estudiarán en el Apartado 9.6.

9.2.3. Creación de valores de tipos complejos

En SQL:1999 se usan las **funciones constructoras** para crear valores de tipos estructurados. Una función con el mismo nombre que un tipo estructurado es una función constructora para el tipo estructurado. Por ejemplo, se podría declarar una constructora para el tipo *Editorial* como:

```
create function Editorial (n varchar(20), s varchar(20))
returns Editorial
begin
    set nombre = n;
    set sucursal = s;
end
```

Se puede usar entonces *Editorial*('McGraw-Hill', 'Nueva York') para crear un valor del tipo *Editorial*.

SQL:1999 también soporta otras funciones además de las constructoras, como se verá en el Apartado 9.6; los nombres de estas funciones deben ser diferentes de cualquier tipo estructurado.

Nótese que en SQL:1999, a diferencia de en las bases de datos orientadas a objetos, un constructor crea un valor del tipo, no un objeto del tipo. Es decir, el valor que crea el constructor no tiene identidad de objeto. Los objetos en SQL:1999 se corresponden con tuplas de una relación, y se crean insertando tuplas en las relaciones.

De manera predeterminada, cada tipo estructurado tiene un constructor sin argumentos, que establece los atributos a sus valores predeterminados. Cualquiera otra constructora tiene que crearse explícitamente. Puede haber más de una constructora para el mismo tipo estructurado; aunque tengan el mismo nombre, tienen que ser distinguibles por el número de argumentos y sus tipos.

En SQL:1999 se puede crear un *array* de valores como:

```
array['Silberschatz', 'Korth', 'Sudarsan']
```

Se puede construir un valor de fila listando sus atributos entre paréntesis. Por ejemplo, si se declara un atributo *editorial1* como un tipo fila (como en el Apartado 9.2.2), se puede construir el siguiente valor para él:

```
('McGraw-Hill', 'Nueva York')
```

sin usar una constructora.

Los atributos de tipo conjunto, tales como *lista-palabras-clave*, se crean enumerando sus elementos entre paréntesis siguiendo a la palabra clave **set**. Se pueden crear valores de tipo multiconjunto al igual que con los valores de tipo conjunto, reemplazando **set** por **multiset**[3].

Así, se puede crear una tupla del tipo definido por la relación *libros* como:

```
('Compiladores',array['Gómez', 'Santos'],
Editorial('McGraw-Hill', 'Nueva York'),
set(' traducción, análisis'))
```

Aquí se ha creado un valor para el atributo *Editorial* invocando a la función *constructora* de *Editorial* con argumentos apropiados.

Si se desea insertar la tupla anterior en la relación *libros*, se podría ejecutar la instrucción:

```
insert into libros
values
('Compiladores',array['Gómez', 'Santos'],
Editorial('McGraw-Hill', 'Nueva York'),
set(' traducción, análisis'))
```

[2] En PL/SQL de Oracle, dada una tabla *t*, *t*%**rowtype** denota el tipo de las filas de la tabla. De forma similar, *t.a*%**type** denota el tipo del atributo *a* de la tabla *t*.

[3] Aunque los conjuntos y multiconjuntos no son parte de la norma SQL:1999, las otras constructoras mostradas en este apartado sí lo son. Las versiones futuras de SQL probablemente darán soporte a los conjuntos y multiconjuntos.

9.3. HERENCIA

La herencia puede hallarse en el nivel de los tipos o en el nivel de las tablas. En primer lugar se considerará la herencia de los tipos y después en el nivel de las tablas.

9.3.1. Herencia de tipos

Supóngase que se dispone de la siguiente definición de tipos para las personas:

create type *Persona*
(*nombre* **varchar**(20),
dirección **varchar**(20))

Puede que se desee guardar en la base de datos más información sobre las personas que sean estudiantes y sobre las que sean profesores. Dado que los estudiantes y los profesores también son personas, se puede utilizar la herencia para definir los tipos estudiante y profesor en SQL:1999:

create type *Estudiante*
under *Persona*
(*curso* **varchar**(20),
departamento **varchar**(20))
create type *Profesor*
under *Persona*
(*sueldo* **integer**,
departamento **varchar**(20))

Tanto *Estudiante* como *Profesor* heredan los atributos de *Persona*, es decir, *nombre* y *dirección*. *Estudiante* y *Profesor* se denominan subtipos de *Persona* y ésta, a su vez, es un supertipo de *Estudiante* y de *Profesor*.

Los métodos de un tipo estructurado se heredan por sus subtipos, al igual que los atributos. Sin embargo, un subtipo puede redefinir el efecto de un método declarando de nuevo el método, usando **overriding method** en lugar de **method** en la declaración del método.

Supóngase ahora que se desea guardar la información sobre los ayudantes, que son simultáneamente estudiantes y profesores, quizás incluso en departamentos diferentes. Esto se puede hacer usando la **herencia múltiple**, que se estudió en el Capítulo 8. La norma SQL:1999 no soporta herencia múltiple. Sin embargo, los borradores de la norma sí lo hacían, y aunque la norma final la omitió, versiones futuras de la norma SQL pueden introducirla. Lo que se expone a continuación se basa en los borradores de la norma.

Por ejemplo, si el sistema de tipos permite la herencia múltiple, se puede definir un tipo para los ayudantes de la manera siguiente:

create type *Ayudante*
under *Estudiante*, *Profesor*

Ayudante heredaría todos los atributos de *Estudiante* y de *Profesor*. Surge un problema, sin embargo, dado que los atributos *nombre*, *dirección* y *departamento* se hallan presentes en *Estudiante* y en *Profesor*.

Los atributos *nombre* y *dirección* se heredan en realidad de un origen común, *Persona*. Así que no se provoca ningún conflicto al heredarlos de *Estudiante* y de *Profesor*. Sin embargo, el atributo *departamento* se define de manera separada en *Estudiante* y en *Profesor*. De hecho, los ayudantes pueden ser estudiantes de un departamento y profesores de otro. Para evitar un conflicto entre los dos ejemplares de *departamento* se les puede cambiar el nombre utilizando una instrucción **as** como se muestra en la siguiente definición del tipo *Ayudante*:

create type *Ayudante*
under *Estudiante* **with** (*departamento* **as**
dep-estudiante)
Profesor **with** (*departamento* **as**
dep-profesor)

SQL:1999 sólo soporta herencia única, es decir, un tipo puede heredar de sólo un tipo único; la sintaxis usadas es como en los ejemplos anteriores. La herencia múltiple como en el ejemplo *Ayudante no* está soportada en SQL:1999. La norma SQL:1999 también requiere un campo extra al final de la definición de tipos, cuyo valor es **final** o **not final**. La palabra clave **final** dice que los subtipos no se pueden crear desde el tipo dado, mientras que **not final** dice que se pueden crear.

En SQL, como en la mayor parte de los lenguajes de programación, las entidades deben tener exactamente un *tipo más específico*. Es decir, cada valor debe estar asociado con un tipo específico, denominado **tipo más específico**, cuando se crea. Mediante la herencia también se asocia con cada uno de los supertipos de su tipo más específico. Por ejemplo, supóngase que una entidad tiene el tipo *Persona* y el tipo *Estudiante*. Por tanto, el tipo más específico de la entidad es *Estudiante*, dado que *Estudiante* es un subtipo de *Persona*. Sin embargo, una entidad no puede tener los tipos *Estudiante* y *Profesor*, a menos que tenga un tipo, como *Ayudante*, que sea un subtipo de *Profesor* y de *Estudiante*.

9.3.2. Herencia de tablas

Las subtablas en SQL:1999 se corresponden con la noción del modelo E-R de la especialización y la generalización. Por ejemplo, supóngase que se define la tabla *personas* de la manera siguiente:

create table *persona* **of** *Persona*

Se pueden definir entonces las tablas *estudiantes* y *profesores* como **subtablas** de *persona*:

create table *estudiantes* **of** *Estudiante*
under *persona*
create table *profesores* **of** *Profesor*
under *persona*

Los tipos de las subtablas deben ser subtipos del tipo de la tabla padre. Por tanto, cada atributo presente en *persona* debe estar también presente en las subtablas.

Además, cuando se declaran *estudiantes* y *profesores* como subtablas de *persona*, cada tupla presente en *estudiantes* o *profesores* también están presentes implícitamente en *persona*. Así, si una consulta usa la tabla *persona*, encontrará no sólo las tuplas insertadas directamente en la tabla, sino también las tuplas insertadas en sus subtablas *estudiantes* y *profesores*. Sin embargo, sólo se puede acceder a los atributos que están presentes en *persona*.

Es posible la herencia múltiple con las tablas, como con los tipos. (Nótese, sin embargo, que la herencia múltiple de tablas no se soporta en SQL:1999.) Por ejemplo, se puede crear una tabla del tipo *Ayudante*:

create table *ayudantes*
of *Ayudante*
under *estudiantes, profesores*

Como resultado de la declaración, cada tupla presente en la tabla *ayudantes* también está presente implícitamente en las tablas *profesores* y *estudiantes*, y a su vez en la tabla *persona*.

SQL:1999 permite buscar tuplas que estén en *persona* pero no en sus subtablas usando «**only** *persona*» en lugar de *persona* en la consulta.

Hay algunos requisitos de consistencia para las subtablas. Antes de indicar las restricciones es necesaria una definición: se dice que las tuplas de una subtabla **corresponden** a las tuplas de una tabla padre si tienen los mismo valores para todos los atributos heredados. Así, las tuplas correspondientes representan la misma entidad.

Los requisitos de consistencia para las subtablas son:

1. Cada tupla de la supertabla puede corresponder a lo sumo con una tupla de cada una de sus tablas inmediatas.
2. SQL:1999 tiene una restricción adicional que establece que todas las tuplas que se correspondan se deben derivar de una tupla (insertada en una tabla).

Por ejemplo, sin la primera condición se podrían tener dos tuplas en *estudiantes* (o en *profesores*) correspondiente a la misma persona.

La segunda condición descarta una tupla en *persona* correspondiente a tuplas de *estudiantes estudiantes* y *profesores*, a menos que esas tuplas estén presentes implícitamente porque se insertó una tupla en la tabla *ayudantes*, que es una subtabla de *profesores* y *estudiantes*.

Dado que SQL:1999 no soporta herencia múltiple, la segunda condición realmente impide que una persona sea tanto profesor como estudiante. El mismo problema surgiría si no existiese la subtabla *ayudantes*, incluso si hubiese herencia múltiple. Obviamente sería útil modelar una situación donde una persona pueda ser profesor y estudiante, incluso si no está presente la subtabla común *ayudantes*. Por tanto, puede ser útil eliminar la segunda restricción de consistencia. Se volverá a este tema en el Apartado 9.3.3.

Las subtablas pueden guardarse de manera eficiente sin réplica de todos los campos heredados de una de las dos siguientes formas:

- Cada tabla almacena la clave primaria (que se puede heredar de una tabla padre) y los atributos definidos localmente. Los atributos heredados (aparte de la clave primaria) no hace falta guardarlos y pueden obtenerse mediante una reunión con la supertabla basada en la clave primaria.
- Cada tabla almacena todos los atributos heredados y definidos localmente. Cuando se inserta una tupla se almacena sólo en la tabla en la que se inserta y su presencia se infiere en cada supertabla. El acceso a todos los atributos de una tupla es más rápido, dado que no se requiere una reunión. Sin embargo, en el caso de que no se considere la segunda restricción de integridad —es decir, una entidad se puede representar en dos subtablas sin estar presente en una subtabla común de ambas— esta representación puede resultar en duplicación de información.

9.3.3. Solapamiento de subtablas

La herencia de tipos debe utilizarse con precaución. Una base de datos universitaria puede tener muchos subtipos de *Persona*, como *Estudiante*, *Profesor*, *JugadorDeFútbol*, *CiudadanoExtranjero*, etcétera. *Estudiante* puede a su vez tener subtipos como *EstudianteDeDiplomatura*, *EstudianteDeLicenciatura* y *EstudianteATiempoParcial*. Evidentemente, una persona puede pertenecer a varias de estas categorías simultáneamente. Como se mencionó en el Capítulo 8, a veces se denomina *papel* a cada una de estas categorías.

Para que cada entidad tenga exactamente un tipo más específico habría que crear un subtipo para cada combinación posible de los supertipos. En el ejemplo anterior habría subtipos como *EstudianteDeDiplomaturaExtranjero*, *EstudianteDeLicenciaturaJugadorDeFútbolExtranjero*, etcétera. Lamentablemente, se acabaría con un número enorme de subtipos de *Persona*.

Un enfoque mejor en el contexto de los sistemas de bases de datos es permitir que un objeto tenga varios

tipos, sin que tenga un tipo más específico. Los sistemas relacionales orientados a objetos pueden modelar esta característica utilizando la herencia en el nivel de las tablas, en vez de en el nivel de los tipos, y permitiendo que las entidades estén en más de una tabla simultáneamente.

Por ejemplo, supóngase de nuevo que se tiene el tipo *Persona* con los subtipos *Estudiante* y *Profesor*, y la tabla correspondiente *persona*, con subtablas *profesores* y *estudiantes*. Se puede entonces tener una tupla en *profesores* y otra en *estudiantes* que correspondan a la misma tupla en *persona*.

No hay necesidad de tener un tipo *Ayudante* como subtipo de *Estudiante* y *Profesor*. No es necesario crear el tipo *Ayudante* a menos que se desee almacenar atributos extra o redefinir métodos de específicamente para las personas que sean estudiantes y profesores.

Sin embargo, SQL:1999 prohíbe esta situación debido al requisito de consistencia 2 del Apartado 9.3.2. Dado que SQL:1999 no soporta herencia múltiple, no se puede usar la herencia para modelar una situación donde una persona pueda ser estudiante y profesor. Por supuesto se pueden crear tablas separadas para representar la información sin usar la herencia. Habría que añadir restricciones de integridad referencial apropiadas para asegurar que los estudiantes y profesores están representados también en la tabla *persona*.

9.4. TIPOS DE REFERENCIA

Los lenguajes orientados a objetos proporcionan la posibilidad de hacer referencia a los objetos. El atributo de un tipo puede ser una referencia a un objeto de un tipo especificado. Por ejemplo, en SQL:1999 se puede definir un tipo *Departamento*, con campos *nombre* y *director*, que es una referencia al tipo *Persona,* y una tabla *departamentos* de tipo *Departamento*, como sigue:

```
create type Departamento(
    nombre varchar(20),
    director ref(Persona) scope persona
)
create table departamentos of Departamento
```

La referencia en este ejemplo está restringida a tuplas de la tabla *persona*. La restricción de **scope** de una referencia a las tuplas de una tabla es obligatoria en SQL:1999 y hace que las referencias se comporten como claves externas.

Se puede omitir la declaración **scope** *persona* de la declaración de tipos y en su lugar añadirla a la instrucción **create table**.

```
create table departamentos of Departamento
    (director with options scope persona)
```

Para inicializar un atributo referencia es necesario obtener el identificador de la tupla a la que se va a hacer referencia. Se puede obtener el valor del identificador de una tupla mediante una consulta. Así, para crear una tupla con el valor referencia, se puede crear en primer lugar la tupla con una referencia nula y después establecer la referencia:

```
insert into departamentos
values ('Informática', null)
update departamentos
    set director = (select ref(p)
                    from persona as p
                    where nombre = 'Juan')
  where nombre = 'Informática'
```

Esta sintaxis para acceder al identificador de una tupla está basada en la sintaxis de Oracle. SQL:1999 adopta un enfoque diferente, en el que la tabla referenciada debe tener un atributo que almacene el identificador de la tupla. Este atributo, denominado **atributo autorreferencial**, se declara añadiendo la cláusula **ref is** a la instrucción **create table**.

```
create table persona of Persona
ref is ido system generated
```

Donde *ido* es un nombre de atributo, no una palabra clave. La subconsulta anterior podría usar

```
select p.ido
```

en lugar de **select ref**(*p*).

Una alternativa a los identificadores generados por el sistema es permitir a los usuarios generar identificadores. El tipo del atributo autorreferencial se debe especificar como parte de la definición de tipos de la tabla referenciada, y la definición de tabla debe especificar que la referencia la genera el usuario (**user generated**).

```
create type Persona
    (nombre varchar(20),
    dirección varchar(20))
    ref using varchar(20)
create table persona of Persona
    ref is ido user generated
```

Al insertar una tupla en *persona* se debe proporcionar un valor para el identificador:

insert into *persona* **values**
('01284567', 'Juan', 'Plaza Mayor, 1')

Ninguna otra tupla de *persona* o sus supertablas pueden tener el mismo identificador. Se puede entonces usar el valor del identificador al insertar una tupla en *departamentos*, sin necesitar una consulta separada para obtener el identificador.

insert into *departamentos*
values ('Informática', '01284567')

Es posible incluso usar un valor existente de clave primaria como identificador, incluyendo la cláusula **ref from** en la definición de tipos:

create type *Persona*
(*nombre* **varchar**(20) **primary key**,
dirección **varchar**(20))
ref from *nombre*
create table *persona* **of** *Persona*
ref is *ido* **derived**

Nótese que la definición de tabla debe especificar que la referencia es derivada y aún debe especificar un nombre de atributo autorreferencial. Al insertar una tupla en *departamentos*, se puede usar:

insert into *departamentos*
values ('Informática', 'Juan')

9.5. CONSULTAS CON TIPOS COMPLEJOS

En este apartado se presenta una extensión del lenguaje de consulta SQL para trabajar con los tipos complejos. Se puede comenzar por un ejemplo sencillo: averiguar el título y el nombre de la editorial de cada documento. La consulta siguiente lleva a cabo esa tarea:

select *título*, *editorial.nombre*
from *libros*

Obsérvese que se hace referencia al campo *nombre* del atributo compuesto *editorial* utilizando una notación con un punto.

9.5.1. Expresiones de ruta

Las referencias se desreferencian en SQL:1999 con el símbolo –>. Considérese otra vez la tabla *departamentos*. Se puede usar la siguiente consulta para hallar los nombres y direcciones de los directores de todos los departamentos.

select *director–>nombre, director–>dirección*
from *departamentos*

Una expresión como «*director–>nombre*» se denomina una **expresión de ruta**.

Dado que *director* es una referencia a una tupla de la tabla *persona*, el atributo *nombre* en la consulta anterior es el atributo *nombre* de la tupla de la tabla *persona*. Se pueden usar las referencias para ocultar las operaciones reunión; en el ejemplo anterior, sin las referencias, el campo *director* de *departamento* se declararía como clave externa de la tabla *persona*. Para encontrar el nombre y dirección del director de un departamento se necesitaría una reunión explícita de las relaciones *departamentos* y *persona*. El uso de referencias simplifica considerablemente la consulta.

9.5.2. Atributos de tipo colección

Ahora se considera la forma de manejar los atributos de tipo colección. Los *arrays* son el único tipo colección soportado por SQL:1999, pero también se usa la misma sintaxis para los atributos de tipo relación. Una expresión que se evalúe a una colección puede aparecer en cualquier lugar en que aparezca un nombre de relación, tal como en la cláusula **from**, como ilustran los siguientes párrafos. Se usa la tabla *libros* que se definió anteriormente.

Si se desea hallar todos los documentos que tienen las palabras «base de datos» entre sus palabras clave se puede utilizar la consulta siguiente:

select *título*
from *libros*
where «*base de datos*» **in** (**unnest**(*lista-palabras-clave*))

Obsérvese que se ha usado **unnest**(*lista-palabras-clave*) en una posición en la que SQL sin relaciones anidadas habría exigido una subexpresión **select-from-where**.

Si se sabe que un libro en particular tiene tres autores, se podría escribir:

select *array-autores*[1], *array-autores*[2],
array-autores[3]
from *libros*
where *título* = 'Fundamentos de bases de datos'

Ahora supóngase que se desea una relación que contenga parejas de la forma «título, nombre-autor» para cada libro y para cada uno de los autores del mismo. Se puede utilizar la consulta siguiente:

select *B.título*, *A*
from *libros* **as** *B*, **unnest**(*B.array-autores*) **as** *A*

Dado que el atributo *array-autores* de *libros* es un campo de tipo colección, puede utilizarse en una instrucción **from** en la que se espere una relación.

9.5.3. Anidamiento y desanidamiento

La transformación de una relación anidada en una forma con menos (o sin) atributos de tipo relación se denomina **desanidamiento**. La relación *libros* tiene dos atributos, *array-autores* y *lista-palabras-clave*, que son colecciones, y otros dos, *título* y *editorial*, que no lo son. Supóngase que se desea convertir la relación en una sola relación plana, sin relaciones anidadas ni tipos estructurados como atributos. Se puede utilizar la siguiente consulta para llevar a cabo la tarea:

```
select título, A as autor, editorial.nombre
    as nombre-edit, editorial.sucursal
    as sucursal.edit,
    K as palabra-clave
from libros as B, unnest(B.array-autores)
    as A, unnest(B.lista-palabras-clave) as K
```

La variable *B* de la cláusula **from** se declara para que tome valores de *libros*. La variable *A* se declara para que tome valores de los autores en *array-autores* para el documento *B* y *K* se declara para que tome valores de las palabras clave de la *lista-palabras-clave* del mismo. En la Figura 9.1 (del Apartado 9.1) se muestra un ejemplo de relación *libros* y la Figura 9.2 muestra la relación 1FN resultado de la consulta anterior.

El proceso inverso de transformar una relación 1FN en una relación anidada se denomina **anidamiento**. El anidamiento puede realizarse mediante una extensión de la agrupación en SQL. En el uso normal de la agrupación en SQL se crea (lógicamente) una relación multiconjunto temporal para cada grupo y se aplica una función de agregación a esa relación temporal. Devolviendo el multiconjunto en lugar de aplicar la función de agregación se puede crear una relación anidada. Supóngase que se tiene la relación 1FN *libros-planos*, tal y como se muestra en la Figura 9.2. La consulta siguiente anida la relación en el atributo *palabra-clave*:

```
select título, autor, Editorial(nombre-edit, sucursal-edit)
    as editorial, set(palabra-clave)
    as lista-palabras-clave
from libros-planos
groupby título, autor, editorial
```

El resultado de la consulta a la relación *libros* de la Figura 9.2 se muestra en la Figura 9.4. Si se desea anidar también el atributo autor y volver a convertir, por tanto, la tabla *libros-planos*, en 1FN, en la tabla anidada *libros* mostrada en la Figura 9.1 se puede utilizar la consulta siguiente:

```
select título, set(autor) as array-autores, Editorial
        (nombre-edit, sucursal-edit) as editorial,
        set(palabra-clave) as lista-palabras-clave
from libros-planos
groupby título, editorial
```

Otro enfoque para la creación de relaciones anidadas es usar subconsultas en la cláusula **select**. La siguiente consulta, que ilustra este enfoque, realiza la misma tarea que la consulta anterior.

```
select título,
    (select autor
     from libros-planos as M
     where M.título = O.título) as lista-autores,
    Editorial(nombre-edit, sucursal-edit) as editorial,
    (select palabra-clave
     from libros-planos as N
     where N.título = O.título) as lista-palabras-clave,
from libros-planos as O
```

El sistema ejecuta las subconsultas anidadas de la cláusula **select** para cada tupla generada por las cláusulas **from** y **where** de la consulta externa. Obsérvese que el atributo *O.título* de la consulta externa se usa en las consultas anidadas para asegurar que sólo se generan los conjuntos correctos de autores y palabras clave para cada título. Una ventaja de este enfoque es que se puede usar una cláusula **order by** en la consulta anidada para generar los resultados en el orden deseado. Sin dicho orden, los *arrays* y las listas no estarían unívocamente determinados.

Mientras que el desanidamiento de los atributos de tipo *array* se puede llevar a cabo en SQL:1999 como se ha mostrado, el proceso inverso de anidamiento no se soporta en SQL:1999. Las extensiones que se han mostrado para el anidamiento ilustran las características de algunas propuestas para la extensión de SQL, pero actualmente no forman parte de la norma.

título	*autor*	*editorial* (*nombre-edit, sucursal-edit*)	*lista-palabras-clave*
Compiladores	Gómez	(McGraw-Hill, Nueva York)	{traducción, análisis}
Compiladores	Santos	(McGraw-Hill, Nueva York)	{traducción, análisis}
Redes	Santos	(Oxford, Londres)	{Internet, Web}
Redes	Escudero	(Oxford, Londres)	{Internet, Web}

FIGURA 9.4. Una versión parcialmente anidada de la relación *libros-planos*.

9.6. FUNCIONES Y PROCEDIMIENTOS

SQL:1999 permite la definición de funciones, procedimientos y métodos. Se pueden definir mediante el componente procedimental de SQL:1999 o mediante un lenguaje de programación como Java, C o C++. En primer lugar se examinarán las definiciones en SQL:1999 y después se verá cómo usar las definiciones en lenguajes externos. Algunos sistemas de bases de datos soportan sus propios lenguajes procedimentales, tales como PL/SQL en Oracle y TransactSQL en SQL Server de Microsoft. Éstos incorporan una parte procedimental parecida a SQL:1999, pero hay diferencias en la sintaxis y la semántica; véanse los manuales de sistema respectivos para más detalles.

9.6.1. Funciones y procedimientos en SQL

Supóngase que se desea una función que, dado un libro, devuelva el recuento del número de autores usando el esquema 4FN. Se puede definir la función de la manera siguiente:

```
create function recuento-autores (título varchar(20))
    returns integer
    begin
    declare recuento-a integer;
        select count (autor) into recuento-a
        from autores
        where autores.título = título
    return recuento-a;
    end
```

La función anterior se puede utilizar en una consulta que devuelva los títulos de todos los libros que tengan más de un autor:

```
select título
from libros4
where recuento-autores(título) > 1
```

Las funciones son particularmente útiles con tipos de datos especializados tales como las imágenes y los objetos geométricos. Por ejemplo, un tipo de datos polígono usado en una base de datos cartográfica puede tener asociada una función que compruebe si solapan dos polígonos, y un tipo de datos imagen puede tener asociadas funciones para comparar la semejanza de dos imágenes. Las funciones se pueden escribir en un lenguaje externo como C, como se vio en el Apartado 9.6.2. Algunos sistemas de bases de datos también soportan funciones que devuelven relaciones, es decir, multiconjuntos de tuplas, aunque tales funciones no se soportan en SQL:1999.

Los métodos, que se vieron en el Apartado 9.2.2, se pueden ver como funciones asociadas a tipos estructurados. Tienen un primer parámetro implícito denominado **self**, que se establece al valor del tipo estructurado sobre el que se invoca el método. Así, el cuerpo del método puede referirse a un atributo *a* del valor usando **self**.*a*. El método también puede actualizar estos atributos.

SQL:1999 también soporta procedimientos. La función *recuento-autores* se podría haber escrito como un procedimiento:

```
create procedure proc-recuento-autores
(in título varchar(20), out recuento-a integer)
    begin
        select count (autor) into recuento-a
        from autores
        where autores.título = título
    end
```

Los procedimientos se pueden invocar desde un procedimiento SQL o desde SQL incorporado con la instrucción **call**:

```
declare recuento-a integer;
call proc-recuento-autores('Fundamentos de bases
de datos', recuento-a);
```

SQL:1999 permite que más de un procedimiento tenga el mismo nombre mientras que el número de los argumentos de estos procedimientos sea diferente. El nombre, junto con el número de argumentos, se usa para identificar el procedimiento. SQL:1999 también permite que más de una función tenga el mismo nombre, siempre que las funciones con el mismo nombre tengan un número diferente de argumentos o que las que tengan el mismo número difieran al menos en el tipo de un argumento.

9.6.2. Rutinas externas del lenguaje

SQL:1999 permite definir funciones en un lenguaje de programación tal como C o C++. Las funciones definidas así pueden ser más eficientes que las definidas en SQL, y los cálculos que no se pueden realizar en SQL se pueden ejecutar por estas funciones. Un ejemplo de tales funciones sería realizar un cálculo aritmético complejo sobre los datos de una tupla.

Los procedimientos y funciones externos se pueden especificar de la siguiente forma:

```
create procedure proc-recuento-autores (in título
  varchar(20), out recuento-a integer)
language C
external name '/usr/avi/bin/proc-recuento-autores'

create function recuento-autores (título varchar(20))
returns integer
language C
external name '/usr/avi/bin/recuento-autores'
```

Los procedimientos externos del lenguaje necesitan manejar valores nulos y excepciones. Deben tener por tanto varios parámetros extra: un valor **sqlstate** para indicar el estado de fallo o éxito, un parámetro para almacenar el valor devuelto por la función, y variables indicadoras para cada parámetro y resultado de la función para indicar si el valor es nulo. La línea extra **parameter style general** añadida a la declaración anterior indica que las funciones o procedimientos externos sólo pueden tomar los argumentos mostrados y no manejan valores nulos o excepciones.

Las funciones definidas en un lenguaje de programación y compiladas fuera del sistema de bases de datos se pueden cargar y ejecutar con el código del sistema de bases de datos. Sin embargo, al hacerlo así se cae en el riesgo de que un error en el programa pueda corromper las estructuras internas de la base de datos, y puede omitir la funcionalidad de control de acceso del sistema de bases de datos. Los sistemas de bases de datos que están más preocupados por el rendimiento que por la seguridad pueden ejecutar procedimientos de esta forma.

Los sistemas de bases de datos que están preocupados por la seguridad ejecutarían normalmente este código como parte de un proceso separado, le comunicarían los valores de los parámetros y extraerían los resultados mediante comunicación entre procesos.

Si el código se escribe en un lenguaje como Java, hay una tercera posibilidad: la ejecución del código en un «*cubo*» dentro del mismo proceso de la base de datos. El cubo evita que el código Java realice cualquier lectura o escritura directamente en la base de datos.

9.6.3. Constructoras procedimentales

SQL:1999 soporta varias constructoras procedimentales que proporcionan casi toda la potencia de un lenguaje de programación de propósito general. La parte de la norma SQL:1999 que maneja estas constructoras se denomina **Módulo de almacenamiento persistente** (Persistent Storage Module, PSM).

Una instrucción compuesta es de la forma **begin** ... **end**, y puede contener varias instrucciones SQL entre **begin** y **end**. Se pueden declarar variables locales dentro de una instrucción compuesta, como se ha visto en el Apartado 9.6.1.

SQL:1999 soporta instrucciones **while** y **repeat** con esta sintaxis:

```
declare n integer default 0;
while n < 10 do
    set n = n +1;
end while
repeat
    set n = n-1;
until n = 0
end repeat
```

Este código no hace nada útil; simplemente está para mostrar la sintaxis de los bucles **while** y **repeat**. Se verán usos con más sentido posteriormente.

También hay un bucle **for**, que permite la iteración sobre los resultados de una consulta.

```
declare n integer default 0;
for r as
    select saldo from cuenta
    where nombre-sucursal = 'Navacerrada'
do
    set n = n + r.saldo;
end for
```

El programa abre implícitamente un cursor cuando el bucle **for** inicia la ejecución y lo usa para extraer los valores por filas en la variable del bucle **for** (*r* en el ejemplo). Es posible dar un nombre al cursor insertando el texto *nc* **cursor for** justo después de la palabra clave **as**, donde *nc* es el nombre que se desea dar al cursor. El nombre del cursor se puede usar para realizar operaciones de actualización o borrado sobre la tupla apuntada por el cursor. La instrucción **leave** se puede usar para abandonar el bucle, mientras que **iterate** comienza en la siguiente tupla, desde el principio del bucle, saltando el resto de instrucciones.

El resto de instrucciones soportadas por SQL:1999 incluye instrucciones if-then-else con esta sintaxis:

```
if r.saldo < 1000
    then set p = p + r.saldo
elseif r.saldo < 5000
    then set m = m + r.saldo
else set g = g + r.saldo
end if
```

Este código asume que *l*, *m* y *h* son variables enteras y que *r* es una variable de filas. Si se reemplaza la línea «**set** *n = n + r.saldo*» en el bucle **for** del párrafo anterior por código **if-then-else**, el bucle calculará los saldos totales de las cuentas que se encuentran por debajo de las categorías de saldo pequeño, medio y grande, respectivamente.

SQL:1999 también da soporte a una instrucción **case** similar a la del lenguaje C/C++ (además de las expresiones **case** que se vieron en el Capítulo 4).

Finalmente, SQL:1999 incluye el concepto de la emisión de **condiciones de excepción**, y la declaración de **controladores** que pueden manejar la excepción, como en el código:

```
declare sin-existencias condition
declare exit handler for sin-existencias
begin
...
end
```

Las instrucciones entre **begin** y **end** pueden emitir una excepción ejecutando **signal** *sin-existencias*. El controlador dice que si surge la condición, la acción a emprender es salir de la instrucción de grupo **begin** ... **end**. Las

acciones alternativas serían **continue**, que continúa la ejecución desde la siguiente instrucción a la que emitió la excepción. Además de condiciones definidas explícitamente, también hay condiciones predefinidas tales como **sqlexception**, **sqlwarning** y **not found**.

La Figura 9.5 proporciona un ejemplo mayor del uso de las constructoras procedimentales de SQL:1999. El procedimiento *hallarEmpl* calcula el conjunto de todos los empleados directos e indirectos de un jefe dado (especificado por el parámetro *jef*) y almacena los nombres de empleados resultantes en una relación llamada *empl*, que se asume que ya está disponible. El conjunto de todos los empleados directos e indirectos es básicamente el cierre transitivo de la relación *jefe*. Se vio cómo expresar tal consulta mediante recursión en el Capítulo 5 (Apartado 5.2.6).

El procedimiento usa dos tablas temporales, *nuevoemp* y *temp*. El procedimiento inserta todos los empleados que trabajan directamente para *jef* antes del bucle **repeat**. Este bucle añade en primer lugar todos los empleados de *nuevoemp* a *empl*. A continuación determina los empleados que trabajan para los de *nuevoemp*, excepto los que ya se hayan determinado que son empleados de *jef*, y los almacena en la tabla temporal *temp*. Finalmente, reemplaza los contenidos de *nuevoemp* por los contenidos de *temp*. El bucle **repeat** termina cuando no se encuentran nuevos empleados (indirectos).

El uso de la cláusula **except** en el procedimiento asegura que éste funciona incluso en el caso (anormal) de que haya un ciclo en los jefes. Por ejemplo, si *a* trabaja para *b*, *b* trabaja para *c* y *c* trabaja para *a*, hay un ciclo.

Mientras que los ciclos no son realistas en el caso real de los jefes, son posibles en otras aplicaciones. Por ejemplo, supóngase que se tiene una relación *vuelo*(*a*, *desde*) que especifica las ciudades a las que se puede llegar con un vuelo directo. Se puede modificar el procedimiento *HallarEmpl* para determinar todas las ciudades que se pueden alcanzar mediante una secuencia de uno o más vuelos desde una ciudad determinada. Todo lo que hay que hacer es reemplazar *jefe* por *vuelo* y reemplazar los nombres de atributo adecuadamente. En esta situación puede haber ciclo por alcanzabilidad, pero el procedimiento funcionaría correctamente, dado que eliminaría las ciudades que ya se han visitado.

```
create procedure hallarEmpl (in jef char(10))
– – Halla todos los empleados que trabajan directa o indirectamente para jef
– – y los añade a la relación empl(nombre).
– – La relación jefe(nombreemp, nombrejef) especifica quién trabaja
– – para quién.
begin
    create temporary table nuevoemp (nombre char(10));
    create temporary table temp (nombre char(10));
    insert into nuevoemp
        select nombreemp
        from jefe
        where nombrejef = jef
    repeat
        insert into empl
            select nombre
            from nuevoemp;

        insert into temp
            (select jefe.nombreemp
                from nuevoemp, jefe
                where nuevoemp.nombreemp = jefe.nombrejef;
            )
            except (
                select nombreemp
                from empl
            );
        delete from nuevoemp;
        insert into nuevoemp
            select *
            from temp;
        delete from temp;

    until not exists (select * from nuevoemp)
    end repeat;
end
```

FIGURA 9.5. Determinación de todos los empleados de un jefe.

9.7. COMPARACIÓN ENTRE LAS BASES DE DATOS ORIENTADAS A OBJETOS Y LAS BASES DE DATOS RELACIONALES ORIENTADAS A OBJETOS

Ya se han estudiado las bases de datos orientadas a objetos creadas alrededor de los lenguajes de programación persistentes y las bases de datos relacionales orientadas a objetos, que son bases de datos orientadas a objetos construidas sobre el modelo relacional. Ambos tipos de sistemas de bases de datos se hallan en el mercado y los diseñadores de bases de datos deben escoger el tipo de sistema que resulte más adecuado para las necesidades de cada aplicación.

Las extensiones persistentes de los lenguajes de programación y los sistemas relacionales orientados a objetos se han dirigido a mercados diferentes. La naturaleza declarativa y la limitada potencia (comparada con la de los lenguajes de programación) del lenguaje SQL proporcionan una buena protección de los datos respecto de los errores de programación y hace que las optimizaciones de alto nivel, como la reducción de E/S, resulten relativamente sencillas. (La optimización de las expresiones relacionales se trata en el Capítulo 13). Los sistemas relacionales orientados a objetos se dirigen a la simplificación de la realización de los modelos de datos y de las consultas mediante el uso de tipos de datos complejos. Las aplicaciones típicas incluyen el almacenamiento y la consulta de datos complejos, incluyendo los datos multimedia.

Los lenguajes declarativos como SQL, sin embargo, imponen una reducción significativa del rendimiento a ciertos tipos de aplicaciones que se ejecutan principalmente en la memoria principal y realizan gran número de accesos a la base de datos. Los lenguajes de programación persistentes se dirigen a las aplicaciones de este tipo que tienen necesidad de elevados rendimientos. Proporcionan acceso a los datos persistentes con poca sobrecarga y eliminan la necesidad de la traducción de los datos si hay que tratarlos utilizando un lenguaje de programación. Sin embargo, son más susceptibles de deteriorar los datos debido a los errores de programación y no suelen disponer de grandes posibilidades de consulta. Las aplicaciones típicas incluyen las bases de datos de CAD.

Los puntos fuertes de los varios tipos de sistemas de bases de datos pueden resumirse de la manera siguiente:

- **Sistemas relacionales:** tipos de datos sencillos, lenguajes de consulta potentes, protección elevada.
- **Bases de datos orientadas a objetos basadas en lenguajes de programación persistentes:** tipos de datos complejos, integración con los lenguajes de programación, elevado rendimiento.
- **Sistemas relacionales orientados a objetos:** tipos de datos complejos, lenguajes de consulta potentes, protección elevada.

Estas descripciones son válidas en general, pero hay que tener en cuenta que algunos sistemas de bases de datos no respetan estas fronteras. Por ejemplo, algunos sistemas de bases de datos orientados a objetos construidos alrededor de lenguajes de programación persistentes se implementan sobre sistemas de bases de datos relacionales. Puede que estos sistemas proporcionen menor rendimiento que los sistemas de bases de datos orientados a objetos construidos directamente sobre los sistemas de almacenamiento, pero proporcionan en parte las garantías de protección más estrictas propias de los sistemas relacionales.

Muchos sistemas de bases de datos relacionales orientados a objetos se construyen sobre bases de datos relacionales existentes. Para ello, los tipos de datos complejos soportados por los sistemas relacionales orientados a objetos necesitan traducirse al sistema de tipos más sencillo de las bases de datos relacionales.

Para comprender cómo se realiza la traducción sólo es necesario examinar la forma en que algunas características del modelo E-R se traducen en relaciones. Por ejemplo, los atributos multivalorados del modelo E-R se corresponden con los atributos de tipo conjunto del modelo relacional orientado a objetos. Los atributos compuestos se corresponden grosso modo con los tipos estructurados. Las jerarquías ES del modelo E-R se corresponden con la herencia de tablas en el modelo relacional orientado a objetos. Las técnicas para convertir las características del modelo E-R a tablas, que se vieron en el Apartado 2.9, se pueden usar con algunas extensiones para traducir los datos relacionales orientados a objetos en datos relacionales.

9.8. RESUMEN

- El modelo de datos relacional orientado a objetos extiende el modelo de datos relacional proporcionando un sistema de tipos enriquecido que incluye tipos colección y orientación a objetos.
- La orientación a objetos proporciona herencia con subtipos y subtablas, así como referencias a objetos (tuplas).
- Los tipos colección incluyen relaciones anidadas, conjuntos, multiconjuntos y *arrays*, y el modelo relacional orientado a objetos permite que los atributos de las tablas sean colecciones.
- La norma SQL:1999 extiende el lenguaje de definición de datos, así como el lenguaje de consultas, y

en particular da soporte a atributos de tipo colección, herencia y referencias a tuplas. Estas extensiones intentan preservar los fundamentos relacionales —en particular, el acceso declarativo a los datos— a la vez que se extiende la potencia de modelado.

- Los sistemas relacionales orientados a objetos (es decir, sistemas de bases de datos basados en el modelo relacional orientado a objetos) proporcionan un camino de migración adecuado para los usuarios de las bases de datos relacionales que desean usar las características de la orientación a objetos.
- También se han descrito extensiones procedimentales proporcionadas por SQL:1999.
- Se han discutido las diferencias entre los lenguajes de programación persistente y los sistemas relacionales orientados a objetos, y se han mencionado criterios para escoger entre ellos.

TÉRMINOS DE REPASO

- Alcance de una referencia.
- Anidamiento y desanidamiento.
- *Arrays*.
- Atributo autorreferencial.
- Conjuntos.
- Constructoras.
- Constructoras procedimentales.
- Controladores.
- Excepciones.
- Expresiones de ruta.
- Funciones y procedimientos en SQL.
- Herencia.
  - Herencia múltiple.
  - Herencia simple.
- Herencia de tablas.
- Herencia de tipos.
- Métodos.
- Modelo relacional anidado.
- Multiconjuntos.
- Objetos grandes de caracteres (clob).
- Objetos grandes en binario (blob).
- Relaciones anidadas.
- Rutinas externas del lenguaje.
- Solapamiento de subtablas.
- Subtabla.
- Tipo más específico.
- Tipos colección.
- Tipos complejos.
- Tipos estructurados.
- Tipos fila.
- Tipos de objetos grandes.
- Tipos referencia.

EJERCICIOS

9.1. Considérese el esquema de la base de datos

Emp = (*nombree*, **setof**(*Hijos*), **setof**(*Conocimientos*))
Hijos = (*nombre*, *Cumpleaños*)
Cumpleaños = (*día*, *mes*, *año*)
Conocimientos = (*escribir-a-máquina*, **setof**(*Exámenes*))
Exámenes = (*año*, *ciudad*)

Asúmase que los atributos de tipo **setof**(*Hijos*), **setof**(*Conocimientos*) y **setof**(*Exámenes*) tienen nombres de atributo *ConjuntoHijos*, *ConjuntoConocimientos* y *ConjuntoExámenes*, respectivamente. Supóngase que la base de datos contiene una relación *emp*(*Emp*). Escríbanse en SQL:1999 las consultas siguientes (con las extensiones descritas en este capítulo).

a. Hallar los nombres de todos los empleados que tengan hijos nacidos en marzo.

b. Hallar los empleados que hicieron un examen del tipo de conocimiento «escribir-a-máquina» en la ciudad «San Rafael».

c. Indicar todos los tipos de conocimiento de la relación *emp*.

9.2. Vuélvase a diseñar la base de datos del Ejercicio 9.1 en la primera forma normal y en la cuarta forma normal. Indíquense las dependencias funcionales o multivaloradas que se den por supuestas. Indíquense también todas las restricciones de integridad referencial que deban incluirse en los esquemas de la primera y de la cuarta formas normales.

9.3. Considérense los esquemas de la tabla *persona* y las tablas *estudiantes* y *profesores* que se crearon bajo *persona* en el Apartado 9.3. Dese un esquema relacional en la tercera forma normal que represente la misma información. Recuérdense las restricciones de las sub-

tablas y dense todas las restricciones que deban imponerse en el esquema relacional para que cada ejemplar de la base de datos del esquema relacional pueda representarse también mediante un ejemplar del esquema con herencia.

9.4. Una compañía de alquiler de coches tiene una base de datos de vehículos con todos los vehículos de su flota actual. Para todos los vehículos incluye su número de bastidor, su número de matrícula, el fabricante, el modelo, la fecha de adquisición y su color. Se incluyen datos específicos para algunos tipos de vehículos:

- Camiones: capacidad de carga.
- Coches deportivos: potencia, edad mínima del arrendatario.
- Monovolúmenes: número de plazas.
- Vehículos todoterreno: altura de los bajos, eje motor (tracción a dos ruedas o a las cuatro).

Constrúyase una definición de esquema para esta base de datos en SQL:1999. Utilícese la herencia donde resulte conveniente.

9.5. Explíquese la diferencia entre un tipo *x* y un tipo referencia **ref**(*x*). ¿En qué circunstancias se debe escoger un tipo referencia?

9.6. Considérese el diagrama E-R de la Figura 2.11, que contiene atributos compuestos, multivalorados y derivados.

a. Dese una definición de esquema en SQL:1999 correspondiente al diagrama E-R. Utilícese un *array* para representar el atributo multivalorado y constructoras apropiadas de SQL:1999 para representar los otros tipos de atributos.

b. Dense constructores para cada uno de los tipos estructurados definidos.

9.7. Dese una definición de esquema en SQL:1999 del diagrama E-R de la Figura 2.17, que contiene especializaciones.

9.8. Considérese el esquema relacional de la Figura 3.39.

a. Dese una definición de esquema en SQL:1999 correspondiente al esquema relacional, pero usando referencias para expresar las relaciones de claves externas.

b. Escríbanse cada una de las consultas del Ejercicio 3.10 en el esquema anterior usando SQL:1999.

9.9. Considérese una base de datos de empleados con las relaciones

empleado(<u>*nombre-empleado*</u>, *calle*, *ciudad*)
trabaja(<u>*nombre-empleado*</u>, *nombre-empresa*, *sueldo*)

donde las claves primarias se han subrayado. Escríbase una consulta para hallar las empresas cuyos empleados ganan un sueldo mayor, de media, que el sueldo medio del Banco Importante.

a. Usando funciones SQL:1999 donde sea apropiado.

b. Sin usar funciones SQL:1999.

9.10. Reescríbase la consulta del Apartado 9.6.1 que devuelve los títulos de todos los libros que tengan más de un autor usando la cláusula **with** en lugar de la función.

9.11. Compárese el uso de SQL incorporado con el uso en SQL de las funciones definidas utilizando un lenguaje de programación de propósito general ¿En qué circunstancias se debe utilizar cada una de estas características?

9.12. Supóngase que se ha sido contratado como asesor para escoger un sistema de bases de datos para la aplicación del cliente. Para cada una de las aplicaciones siguientes indíquese el tipo de sistema de bases de datos (relacional, base de datos orientada a objetos basada en un lenguaje de programación persistente, relacional orientada a objetos; no se debe especificar ningún producto comercial) que se recomendaría. Justifíquese la recomendación.

a. Sistema de diseño asistido por computadora para un fabricante de aviones.

b. Sistema para realizar el seguimiento de los donativos hechos a los candidatos a un cargo público.

c. Sistema de información de ayuda para la realización de películas.

NOTAS BIBLIOGRÁFICAS

El modelo relacional anidado lo introdujeron Makinouchi [1977] y Jaeschke y Schek [1982]. En Fischer y Thomas [1983], Zaniolo [1983], Ozsoyoglu et al. [1987], Gucht [1987] y Roth et al. [1988] se presentan varios lenguajes algebraicos de consulta. La gestión de los valores nulos en las relaciones anidadas se discute en Roth et al. [1989]. Los problemas de diseño y de normalización se discuten en Ozsoyoglu y Yuan [1987], Roth y Korth [1987] y Mok et al. [1996]. En Abiteboul et al. [1989] aparece una colección de trabajos sobre las relaciones anidadas.

Se han propuesto varias extensiones de SQL orientadas a objetos. POSTGRES (Stonebraker y Rowe [1986] y Stonebraker [1986a, 1987]) fue una de las primeras implementaciones de un sistema relacional orientado a objetos. Illustra es el sistema relacional orientado a objetos comercial sucesor de POSTGRES (Informix compró posteriormente Illustra, que a su vez fue comprado por IBM). El sistema de bases de datos Iris de Hewlett-Packard (Fishman et al. [1990] y Wilkinson et al. [1990]) proporciona extensiones orientadas a objetos sobre un sistema de bases de datos relacional. El lenguaje de consulta O_2 descrito en Bancilhon et al. [1989] es una extensión de SQL orientada a objetos implementada en el sistema de bases de datos orientado a objetos O_2 (Deux [1991]). UniSQL se describe en

UniSQL [1991]. XSQL es una extensión de SQL orientada a objetos propuesta por Kifer et al. [1992].

SQL:1999 fue el producto de un esfuerzo extensivo (y retrasado) de normalización, que se inició originalmente añadiendo características de la programación orientada a objetos a SQL, y finalizó añadiéndose muchas otras características, como el flujo de control, como se ha visto. Los documentos de la norma están disponible (mediante pago) en http://webstore.ansi.org. Sin embargo, los documentos de la norma son difíciles de leer y se dejan a los implementadores de SQL:1999. Hay libros en edición sobre SQL:1999 en el momento de la escritura de este libro; véase el sitio Web del libro para información actual.

HERRAMIENTAS

El sistema de bases de datos Informix proporciona soporte para muchas características relacionales orientadas a objetos. Oracle introdujo varias características relacionales orientadas a objetos en Oracle 8.0. Ambos sistemas proporcionaron características relacionales orientadas a objetos antes de que la norma SQL:1999 estuviese finalizada, y tienen algunas características que no forman parte de la norma SQL:1999. DB2 de IBM soporta muchas de las características de SQL:1999.

CAPÍTULO

10 XML

A diferencia de la mayor parte de las tecnologías presentadas en los capítulos anteriores, el **lenguaje de marcas extensible (*Extensible Markup Language, XML*)** no se concibió como una tecnología para bases de datos. En realidad, al igual que el *lenguaje de marcas de hipertexto* (*Hyper-Text Markup Language,* HTML) sobre el que está basado World Wide Web, XML; tiene sus raíces en la gestión de documentos y está derivado de un lenguaje para estructurar documentos grandes conocido como *lenguaje estándar generalizado de marcas* (*Standard Generalized Markup Language,* SGML). Sin embargo, a diferencia de SGML y HTML, XML puede representar datos de bases de datos, así como muchas clases de datos estructurados usadas en aplicaciones de negocios. Es particularmente útil como formato de datos cuando las aplicaciones se deben comunicar con otra aplicación o integrar información de varias aplicaciones. Cuando XML se usa en estos contextos, se generan muchas dudas sobre las bases de datos, incluyendo cómo organizar, manipular y consultar los datos XML. En este capítulo se realiza una introducción sobre XML y se discute la gestión de los datos XML con las técnicas de bases de datos, así como el intercambio de datos con formato como documentos XML.

10.1. ANTECEDENTES

Para comprender XML es importante entender sus raíces como un lenguaje de marcas de documentos. El término **marca** se refiere a cualquier elemento en un documento del que no se tiene intención que sea parte de la salida impresa. Por ejemplo, un escritor que crea un texto que finalmente se compone en una revista puede desear realizar notas sobre cómo se ha de realizar la composición. Sería importante introducir estas notas de forma que se pudieran distinguir del contenido real, de forma que una nota como «no romper esta párrafo» no acabe impresa en la revista. En un procesamiento electrónico de documentos un **lenguaje de marcas** es una descripción formal de qué parte del documento es contenido, qué parte es marca y lo que significa la marca.

Así como los sistemas de bases de datos evolucionaron desde el procesamiento físico de archivos para proporcionar una vista lógica aislada, los lenguajes de marcas evolucionaron desde la especificación de instrucciones que indicaban cómo imprimir partes del documento para la *función* del contenido. Por ejemplo, con marcas funcionales, el texto que representa los encabezamientos de sección (para esta sección la palabra «Antecedentes») se marcaría como un encabezamiento de sección en lugar de marcarse como un texto con el fin de ser impreso en un tamaño agrande, con fuente en negrita. Dicha marca funcional permite que el documento tenga distintos formatos en situaciones diferentes. También ayuda a que distintas partes de un documento largo, o distintas páginas en un sitio Web grande, tengan un formato uniforme. La marca funciona también ayuda a la extracción automática de partes claves de los documentos.

Para la familia de lenguajes de marcado, en los que se incluye HTML, SGML y XML las marcas adoptan la forma de **etiquetas** encerradas entre corchetes angulares , < >. Las etiquetas se usan en pares, con <etiqueta> y </etiqueta> delimitando el comienzo y final de la porción de documento a la cual se refiere la etiqueta. Por ejemplo, el título de un documento podría estar marcado de la siguiente forma:

```
<title>Fundamentos de bases de datos</title>
```

A diferencia de HTML, XML no prescribe las etiquetas permitidas, y se pueden establecer etiquetas según cada necesidad. Esta característica es la clave de la función principal de XML en la representación e intercambio de datos, mientras que HTML se usa principalmente para el formato de documentos.

Por ejemplo, en nuestra aplicación del banco, la información de la cuenta y del cliente se puede representar como parte de un documento XML, como se muestra en la Figura 10.1. Obsérvese el uso de etiquetas tales como cuenta y número-cuenta. Estas etiquetas proporcionan el contexto de cada valor y permiten identificar la semántica del valor.

```
</banco>
    <cuenta>
        <número-cuenta> C-101 </número-cuenta>
        <nombre-sucursal> Centro </nombre-sucursal>
        <saldo> 500 </saldo>
    </cuenta>
    <cuenta>
        <número-cuenta> C-102 </número-cuenta>
        <nombre-sucursal> Navacerrada </nombre-sucursal>
        <saldo> 400 </saldo>
    </cuenta>
    <cuenta>
        <número-cuenta> C-201 </número-cuenta>
        <nombre-sucursal> Galapagar </nombre-sucursal>
        <saldo> 900 </saldo>
    </cuenta>
    <cliente>
        <nombre-cliente> González </nombre-cliente>
        <calle-cliente> Arenal </calle-cliente>
        <ciudad-cliente> La Granja </ciudad-cliente>
    </cliente>
    <cliente>
        <nombre-cliente> López </nombre-cliente>
        <calle-cliente> Mayor </calle-cliente>
        <ciudad-cliente> PeguEtrerosos </ciudad-cliente>
    </cliente>
    <impositor>
        <número-cuenta> C-101 </número-cuenta>
        <nombre-cliente> González </nombre-cliente>
    </impositor>
    <impositor>
        <número-cuenta> C-201 </número-cuenta>
        <nombre-cliente> González </nombre-cliente>
    </impositor>
    <impositor>
        <número-cuenta> C-102 </número-cuenta>
        <nombre-cliente> López </nombre-cliente>
    </impositor>
</banco>
```

FIGURA 10.1. Representación XML de información bancaria.

Comparado al almacenamiento de los datos en una base de datos, la representación XML puede parecer poco eficiente, puesto que los nombres de las etiquetas se repiten por todo el documento. Sin embargo, a pesar de esta desventaja, una representación XML presenta ventajas significativas cuando se usa para el intercambio de datos como, por ejemplo, parte de un mensaje.

- En primer lugar la presencia de las etiquetas hace que el mensaje sea **autodocumentado**, es decir, no se tiene que consultar un esquema para comprender el significado del texto. Se puede leer fácilmente el fragmento anterior, por ejemplo.
- En segundo lugar, el formato del documento no es rígido. Por ejemplo, si algún remitente agrega información adicional tal como una etiqueta último-acceso que informa de la última fecha en la que se ha accedido a la cuenta, el receptor de los datos XML puede sencillamente ignorar la etiqueta. La habilidad de reconocer e ignorar las etiquetas inesperadas permite al formato de los datos evolucionar con el tiempo sin invalidar las aplicaciones existentes.
- Finalmente, puesto que el formato XML está ampliamente aceptado hay una gran variedad de herramientas disponibles para ayudar a su procesamiento, incluyendo software de búsqueda y herramientas de bases de datos.

Al igual que SQL es el lenguaje dominante para consultar los datos relacionales XML se está convirtiendo en el formato dominante para el intercambio de datos.

10.2. ESTRUCTURA DE LOS DATOS XML

El constructor fundamental en un documento XML es el **elemento**. Un elemento es sencillamente un par de etiquetas de inicio y finalización coincidentes y todo el texto que aparece entre ellas.

Los documentos XML deben tener un único elemento **raíz** que abarque el resto de elementos en el documento. En el ejemplo de la Figura 10.1 el elemento <banco> forma el elemento raíz. Además, los elementos en un documento XML se deben anidar adecuadamente. Por ejemplo

```
<cuenta> … <saldo> … </saldo> … </cuenta>
```

está anidado adecuadamente, mientras que

```
<cuenta> … <saldo> … </cuenta> … </saldo>
```

no está adecuadamente anidado.

Aunque el anidamiento adecuado es una propiedad intuitiva, la debemos definir más formalmente. Se dice que el texto aparece **en el contexto de** un elemento si aparece entre la etiqueta de inicio y la etiqueta de finalización de dicho elemento. Las etiquetas están anidadas adecuadamente si toda etiqueta de inicio tiene una única etiqueta de finalización coincidente que está en el contexto del mismo elemento padre.

Nótese que el texto puede estar mezclado con los subelementos de otro elemento, como en la Figura 10.2. Como con otras características de XML, esta libertad tiene más sentido en un contexto de procesamiento de documentos que en el contexto de procesamiento de datos y no es particularmente útil para representar en XML datos más estructurados como son el contenido de las bases de datos.

La capacidad de anidar elementos con otros elementos proporciona una forma alternativa de repre-

```
…
    <cuenta>
        Esta cuenta se usa muy rara vez por no decir nunca.
        <número-cuenta> C-102 </número-cuenta>
        <nombre-sucursal> Navacerrada </nombre-sucursal>
        <saldo> 400 </saldo>
    </cuenta>
…
```

FIGURA 10.2. Mezcla de texto con subelementos.

sentar información. La Figura 10.3 muestra una representación de la información bancaria de la Figura 10.1, pero con los elementos cuenta anidados con los elementos cliente, aunque almacenaría elementos cuenta de una forma redundante si pertenecen a varios clientes.

Las representaciones anidadas se usan ampliamente en las aplicaciones de intercambio de datos XML para evitar las reuniones. Por ejemplo, una aplicación de envíos almacenaría la dirección completa del emisor y receptor de una forma redundante en un documento de envío asociado con cada envío mientras que una representación normalizada puede requerir una reunión de registros de envío con una relación *compañía-dirección* para obtener la información de la dirección.

Además de los elementos, XML especifica la noción de **atributo**. Por ejemplo, el tipo de una cuenta se puede representar como un atributo, como en la Figura 10.4. Los atributos de un elemento aparecen como pares *nombre = valor* antes del cierre «>» de una etiqueta. Los atributos son cadenas y no contienen marcas. Además,

```
<banco-1>
    <cliente>
        <nombre-cliente> González </nombre-cliente>
        <calle-cliente> Arenal </calle-cliente>
        <ciudad-cliente> La Granja </ciudad-cliente>
        <cuenta>
            <número-cuenta> C-101 </número-cuenta>
            <nombre-sucursal> Centro </nombre-sucursal>
            <saldo> 500 </saldo>
        </cuenta>
        <cuenta>
            <número-cuenta> C-201 </número-cuenta>
            <nombre-sucursal> Galapagar </nombre-sucursal>
            <saldo> 900 </saldo>
        </cuenta>
    </cliente>
    <cliente>
        <nombre-cliente> López </nombre-cliente>
        <calle-cliente> Mayor </calle-cliente>
        <ciudad-cliente> PeguEtrerosos </ciudad-cliente>
        <cuenta>
            <número-cuenta> C-102 </número-cuenta>
            <nombre-sucursal> Navacerrada </nombre-sucursal>
            <saldo> 400 </saldo>
        </cuenta>
    </cliente>
</banco-1>
```

FIGURA 10.3. Representación XML anidada de información bancaria.

```
…
    <cuenta tipo-cuenta = «corriente»>
        <número-cuenta> C-102 </número-cuenta>
        <nombre-sucursal> Navacerrada </nombre-sucursal>
        <saldo> 400 </saldo>
    </cuenta>
…
```

FIGURA 10.4. Uso de atributos.

los atributos pueden aparecer solamente una vez en una etiqueta dada, al contrario que los subelementos, que pueden estar repetidos.

Nótese que en un contexto de construcción de un documento es importante la distinción entre subelemento y atributo; un atributo es implícitamente texto que no aparece en el documento impreso o visualizado. Sin embargo, en las aplicaciones de bases de datos y de intercambio de datos de XML esta distinción es menos relevante y la elección de representar los datos como un atributo o un subelemento es frecuentemente arbitraria.

Una nota sintáctica final es que un elemento de la forma <elemento></elemento>, que no contiene subelementos o texto, se puede abreviar como <elemento/>; los elementos abreviados pueden, no obstante, contener atributos.

Puesto que los documentos XML se diseñan para su intercambio entre aplicaciones se tiene que introducir un mecanismo de **espacio de nombres** para permitir a las organizaciones especificar nombre únicos globalmente para que se usen como marcas de elementos en los documentos. La idea de un espacio de nombres es anteponer cada etiqueta o atributo con un identificador de recursos universal (por ejemplo, una dirección Web). Por ello, por ejemplo, si Banco Principal deseara asegurar que los documentos XML creados no duplican las etiquetas usadas por los documentos de otros socios del negocio, se puede anteponer un identificador único con dos puntos a cada nombre de etiqueta. El banco puede usar un URL Web como el siguiente

http://www.BancoPrincipal.com

como un identificador único. El uso de identificadores únicos largos en cada etiqueta puede ser poco conveniente, por lo que el espacio de nombres estándar proporciona una forma de definir una abreviatura para los identificadores.

En la Figura 10.5 el elemento raíz (banco) tiene un atributo xmlns:BP, que declara que BP está definido como abreviatura para el URL dado anteriormente. Se puede usar entonces la abreviatura en varias marcas de elementos como se ilustra en la figura.

Un documento puede tener más de un espacio de nombres, declarado como parte del elemento raíz. Se pueden asociar entonces elementos diferentes con espacios de nombres distintos. Se puede definir un espa-

```
<banco xmlns:BP = «http://www.BancoPrincipal.com»>
    …
    <BP:sucursal>
        <BP:nombresucursal> Centro </BP:nombresucursal>
        <BP:ciudadsucursal> Brooklyn </BP:ciudadsucursal>
    </BP:sucursal>
    …
</banco>
```

FIGURA 10.5. Nombres únicos de etiqueta mediante el uso de espacios de nombres.

cio de nombres predeterminado mediante el uso del atributo xmlns en lugar de xmlns:BP en el elemento raíz. Los elementos sin un prefijo de espacio de nombres explícito pertenecen entonces al espacio de nombres predeterminado.

Algunas veces se necesitan almacenar valores que contengan etiquetas sin que sean interpretadas como etiquetas XML. XML permite esta construcción para ello:

```
<![CDATA[<cuenta> ··· </cuenta>]]>
```

Debido a que el texto <cuenta> está encerrado en CDATA, se trata como datos de texto normal, no como una etiqueta. El término CDATA viene de datos de carácter (*character data* en inglés).

10.3. ESQUEMA DE LOS DOCUMENTOS XML

Las bases de datos tienen esquemas que se usan para restringir qué información se puede almacenar en la base de datos y para restringir los tipos de datos de la información almacenada. En cambio, los documentos XML se pueden crear de forma predeterminada sin un esquema asociado. Un elemento puede tener entonces cualquier subelemento o atributo. Aunque dicha libertad puede ser aceptable algunas veces, dada la naturaleza autodescriptiva del formato de datos, no es útil generalmente cuando los documentos XML se deben procesar automáticamente como parte de una aplicación o incluso cuando se van a dar formato en XML a grandes cantidades de datos relacionados.

Aquí se describirá el mecanismo de esquema orientado a documentos incluido como parte del estándar XML, la definición de tipos de documento, así como XMLSchema, definido más recientemente.

10.3.1. Definición de tipos de documento

La **definición de tipos de documento** (*Document Type Definition*, DTD) es una parte opcional de un documento XML. El propósito principal de DTD es similar al de un esquema: restringir el tipo de información presente en el documento. Sin embargo, DTD no restringe en realidad los tipos en el sentido de tipos básicos como entero o cadena. En su lugar solamente restringe el aspecto de subelementos y atributos en un elemento. DTD es principalmente una lista de reglas que indican el patrón de subelementos que aparecen en un elemento. La Figura 10.6 muestra una parte de una DTD de ejemplo de un documento de información bancaria; el documento XML en la Figura 10.1 se ajusta a DTD.

Cada declaración está en la forma de una expresión normal para los subelementos de un elemento. Así, en la DTD de la Figura 10.6 un elemento bancario consiste en uno o más elementos cuenta, cliente o impositor; el operador | especifica «o» mientras que el operador + especifica «uno o más». Aunque no se muestra aquí, el operador * se usa para especificar «cero o más» mientras que el operador ? se usa para especificar un elemento opcional (es decir, «cero o uno»).

El elemento cuenta se define para contener los subelementos número-cuenta, nombre-sucursal y saldo (en ese orden). De forma similar, cliente e impositor tienen los atributos en su esquema definidos como subelementos.

Finalmente, se declara a los elementos número-cuenta, nombre-sucursal, saldo, nombre-cliente, calle-cliente y ciudad-cliente del tipo #PCDATA. La palabra clave #PCDATA indica dato de texto; deriva su nombre históricamente de *parsed character data* (datos de carac-

```
<!DOCTYPE banco [
    <!ELEMENT banco ( (cuenta|cliente|impositor)+)>
    <!ELEMENT cuenta ( número-cuenta nombre-sucursal saldo )>
    <!ELEMENT cliente ( nombre-cliente calle-cliente ciudad-cliente )>
    <!ELEMENT impositor ( nombre-cliente número-cuenta )>
    <!ELEMENT número-cuenta ( #PCDATA )>
    <!ELEMENT nombre-sucursal ( #PCDATA )>
    <!ELEMENT saldo( #PCDATA )>
    <!ELEMENT nombre-cliente( #PCDATA )>
    <!ELEMENT calle-cliente( #PCDATA )>
    <!ELEMENT ciudad-cliente( #PCDATA )>
] >
```

FIGURA 10.6. Ejemplo de una DTD.

teres analizados). Otros dos tipos especiales de declaraciones son empty (vacío), que dice que el elemento no tiene ningún contenido, y any (cualquiera), que indica que no hay restricción sobre los subelementos del elemento; es decir, cualquier elemento, incluso los no mencionados en la DTD, puede ser subelemento del elemento. La ausencia de una declaración para un subelemento es equivalente a declarar explícitamente el tipo como any.

Los atributos permitidos para cada elemento también se declaran en la DTD. Al contrario que los subelementos no se impone ningún orden a los atributos. Los atributos se pueden especificar del tipo CDATA, ID, IDREF o IDREFS; el tipo CDATA simplemente dice que el atributo contiene datos de caracteres mientras que los otros tres no son tan sencillos; se explicarán detalladamente en breve. Por ejemplo, la siguiente línea de una DTD especifica que el elemento cuenta tiene un atributo del tipo tipo-cuenta con valor predeterminado corriente.

```
<!ATTLIST cuenta tipo-cuenta CDATA «corriente» >
```

Los atributos siempre deben tener una declaración de tipo y una declaración predeterminada. La declaración predeterminada puede consistir en un valor predeterminado para el atributo o #REQUIRED, queriendo esto decir que se debe especificar un valor para el atributo en cada elemento, #IMPLIED, lo que significa que no se ha proporcionado ningún valor predeterminado. Si un atributo tiene un valor predeterminado, para cada elemento que no tenga especificado un valor para el atributo el valor se rellena automáticamente cuando se lee el documento XML.

Un atributo del tipo ID proporciona un identificador único para el elemento; un valor que tiene un atributo ID de un elemento no debe estar presente en ningún otro elemento del mismo documento. A lo sumo se permite que el atributo de un elemento sea del tipo ID.

Un atributo del tipo IDREF es una referencia a un elemento; el atributo debe contener un valor que aparezca en el atributo ID de algún elemento en el documento. El tipo IDREFS permite una lista de referencias, separadas por espacios.

La Figura 10.7 muestra una DTD de ejemplo en la que las relaciones de la cuenta de un cliente se representan mediante los atributos ID e IDREFS en lugar de los registros impositor. Los elementos cuenta usan número-cuenta como su atributo identificador; para realizar esto se ha hecho que número-cuenta sea un atributo de cuenta en lugar de un subelemento. Los elementos cliente tienen un nuevo atributo identificador denominado cliente-ID. Además, cada elemento cliente contiene un atributo cuentas del tipo IDREFS, que es una lista de identificadores de las cuentas que posee el cliente. Cada elemento cuenta tiene un atributo tenedores del tipo IDREFS, que es una lista de propietarios de la cuenta.

La Figura 10.8 muestra un ejemplo de documento XML basado en la DTD de la Figura 10.7.

Nótese que se usa un conjunto distinto de cuentas y clientes del ejemplo anterior con el fin de ilustrar mejor la característica IDREFS.

Los atributos ID e IDREF juegan la misma función que los mecanismos de referencia en las bases de datos orientadas a objetos y las bases de datos relacionales orientadas a objetos permitiendo la construcción de complejas relaciones de datos.

Las definiciones de tipos de documentos están fuertemente conectadas con la herencia del formato del documento XML. Debido a esto no son adecuadas por varios motivos para servir como estructura de tipos de XML para aplicaciones de procesamiento de datos. No obstante, un tremendo número de formatos de intercambio de datos se están definiendo en términos de DTD, puesto que fueron parte original del estándar. Veamos algunas limitaciones de las DTD como mecanismo de esquema.

- No se pueden declarar el tipo de elementos y atributos de texto individuales. Por ejemplo, el elemento saldo no se puede restringir para que sea un número positivo. La falta de tal restricción es problemática para las aplicaciones de procesamiento e intercambio de datos, las cuales deben contener el código para verificar los tipos de los elementos y atributos.
- Es difícil usar el mecanismo DTD para especificar conjuntos desordenados de subelementos. El orden es rara vez importante para el intercambio de datos (al contrario que en el diseño de docu-

```
<!DOCTYPE banco-2 [
    <!ELEMENT cuenta ( sucursal, saldo )>
    <!ATTLIST cuenta
        número-cuenta ID #REQUIRED
        tenedores IDREFS #REQUIRED >
    <!ELEMENT cliente ( nombre-cliente, calle-cliente, ciudad-cliente )>
    <!ATTLIST cliente
        cliente-id ID #REQUIRED
        cuentas IDREFS #REQUIRED >
    ··· declaraciones para sucursal, saldo, nombre-cliente,
        calle-cliente y ciudad-cliente ···
] >
```

FIGURA 10.7. DTD con los tipos de atributo ID e IDREF.

```
<banco-2>
    <cuenta número-cuenta = «C-401» tenedores = «C100 C102»>
        <nombre-sucursal> Centro </nombre-sucursal>
        <saldo> 500 </saldo>
    </cuenta>
    <cuenta número-cuenta = «C-402» tenedores = «C102 C101»>
        <nombre-sucursal> Navacerrada </nombre-sucursal>
        <saldo> 900 </saldo>
    </cuenta>
    <cliente cliente-id = «C100» cuentas = «C-401»>
        <nombre-cliente>Juncal</nombre-cliente>
        <calle-cliente> Mártires </calle-cliente>
        <ciudad-cliente> Melilla </ciudad-cliente>
    </cliente>
    <cliente cliente-id = «C101» cuentas = «C-402»>
        <nombre-cliente>Loreto</nombre-cliente>
        <calle-cliente> Montaña </calle-cliente>
        <ciudad-cliente> Cáceres </ciudad-cliente>
    </cliente>
        <cliente cliente-id = «C102» cuentas = «C-401 C-402»>
        <nombre-cliente>María</nombre-cliente>
        <calle-cliente> Etreros </calle-cliente>
        <ciudad-cliente> Alicante </ciudad-cliente>
    </cliente>
</banco-2>
```

FIGURA 10.8. Datos XML con atributos ID e IDREF.

mentos, donde es crucial). Aunque la combinación de la alternativa (la operación |) y la operación * como en la Figura 10.6 permite la especificación de colecciones desordenadas de marcas, es mucho más complicado especificar que cada marca pueda aparecer solamente una vez.

- Hay una falta de tipos en ID e IDREF. Por ello no hay forma de especificar el tipo de elemento al cual se debería referir una atributo IDREF o IDREFS. Como resultado, la DTD de la Figura 10.7 no evita que el atributo «tenedores» de un elemento cuenta se refiera a otras cuentas, incluso si esto no tiene sentido.

10.3.2. Esquema XML

Un intento de reparar muchas de estas deficiencias de DTD produjo un lenguaje de esquema más sofisticado, XMLSchema. Presentamos aquí un ejemplo de XMLSchema y se listan algunas áreas en las cuales mejora a las DTDs sin dar mucho detalle de la sintaxis de XMLSchema.

La Figura 10.9 muestra cómo la DTD de la Figura 10.6 se puede representar mediante XMLSchema. El primer elemento es el elemento raíz banco, cuyo tipo se declara posteriormente. En el ejemplo se definen después los tipos de los elementos cuenta, cliente e impositor. Obsérvese el uso de los tipos xsd:string y xsd:decimal para restringir los tipos de los elementos de datos. Finalmente, el ejemplo define el tipo TipoBanco para contener cero o más apariciones de cada cuenta, cliente e impositor. XMLSchema puede definir el número mínimo y máximo de apariciones de subelementos

```
<xsd:schema xmlns:xsd = «http://www.w3.org/2001/XMLSchema»>
<xsd:element name = «banco» type = «TipoBanco» />
<xsd:element name = «cuenta»>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name = «número-cuenta» type = «xsd:string»/>
            <xsd:element name = «nombre-sucursal» type = «xsd:string»/>
            <xsd:element name = «saldo» type = «xsd:decimal»/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = «cliente»>
            <xsd:element name = «número-cliente» type = «xsd:string»/>
            <xsd:element name = «calle-cliente» type = «xsd:string»/>
            <xsd:element name = «ciudad-cliente» type = «xsd:string»/>
</xsd:element>
<xsd:element name = «impositor»>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name = «nombre-cliente» type = «xsd:string»/>
            <xsd:element name = «número-cuenta» type = «xsd:string»/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:complexType name = «TipoBanco»>
    <xsd:sequence>
        <xsd:element ref = «cuenta» minOccurs = «0» maxOccurs = «unbounded»/>
        <xsd:element ref = «cliente» minOccurs = «0» maxOccurs = «unbounded»/>
        <xsd:element ref = «impositor» minOccurs = «0» maxOccurs = «unbounded»/>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

FIGURA 10.9. Versión XMLSchema del DTD de la Figura 10.6.

mediante minOccurs y maxOccurs. El valor predeterminado para las apariciones máxima y mínima es 1, por lo que se tiene que especificar explícitamente para permitir cero o más cuentas, impositores y clientes.

Entre los beneficios que ofrece XMLSchema respecto a las DTDs se encuentran los siguientes:

- Permite crear tipos definidos por el usuario
- Permite que el texto que aparece en los elementos esté restringido a tipos específicos tales como tipos numéricos en formatos específicos o incluso tipos más complicados como listas o uniones.
- Permite restringir los tipos para crear tipos especializados, por ejemplo especificando valores mínimo y máximo.
- Permite la extensión de tipos complejos mediante el uso de una forma de herencia.
- Es un superconjunto de DTDs.
- Permite restricciones de unicidad y de clave externa.
- Está integrado con espacios de nombres para permitir a diferentes partes de un documento adaptarse a un esquema diferente.
- Él mismo se especifica mediante sintaxis XML como muestra la Figura 10.9.

Sin embargo, el precio a pagar por estas características es que XMLSchema es significativamente más complicado que las DTDs.

10.4. CONSULTA Y TRANSFORMACIÓN

Dado el creciente número de aplicaciones que usan XML para intercambiar, transmitir y almacenar datos, las herramientas para una gestión efectiva de datos XML están siendo cada vez más importantes. En particular las herramientas para consultar y transformar los datos XML son esenciales para extraer información de grandes cuerpos de datos XML y para convertir los datos entre distintas representaciones (esquemas) en XML. Al igual que la salida de una consulta relacional es una relación, la salida de una consulta XML puede ser un documento XML. Como resultado, la consulta y la transformación se pueden combinar en una única herramienta.

Varios lenguajes proporcionan grados crecientes de capacidades de consulta y transformación:

- XPath es un lenguaje para expresiones de rutas de accesos y es realmente un bloque constructor los dos lenguajes de consulta restantes.
- XSLT fue diseñado para ser un lenguaje de transformación como parte del sistema de hojas de estilo XSL, que se usa para controlar el formato de los datos XML en HTML u otro lenguaje de impresión o visualización. Aunque diseñado para el formato, XSLT puede generar XML como salida y puede expresar muchas consultas interesantes. Además, es actualmente el lenguaje más ampliamente disponible para manipular datos XML.
- XQuery ha sido propuesto como un estándar para consultar datos XML. XQuery combina las características de muchas propuestas anteriores para la consulta de XML, en particular el lenguaje Quilt.

Se usa en todos estos lenguajes un **modelo de árbol** de datos XML. Un documento XML se modela como un **árbol** con **nodos** para los elementos y atributos. Los nodos elemento pueden tener nodos hijo, los cuales pueden ser subelementos o atributos del elemento. De igual forma, cada nodo (ya sea atributo o elemento) distinto del elemento raíz tiene un nodo padre, que es un elemento. El orden de los elementos y atributos en el documento XML se modela ordenando los nodos hijos de un árbol. Los términos padre, hijo, ascendiente, descendiente y hermano se interpretan en el modelo de árbol de datos XML.

El contenido textual de un elemento se puede modelar como un nodo de texto hijo del elemento. Los elementos que contienen texto descompuesto por subelementos intervinientes pueden tener varios nodos de texto hijo. Por ejemplo, un elemento que contenga «Éste es un <bold> buen </bold> libro» contiene un subelemento hijo correspondiente al elemento bold y dos nodos de texto hijo correspondientes a «Éste es un» y «libro». Puesto que dichas estructuras se usan comúnmente en los datos de una base de datos se asumirá que los elementos no contienen texto ni subelementos.

10.4.1. XPath

XPath trata partes de un documento XML mediante expresiones de rutas de acceso. Se puede ver el lenguaje como una extensión a las expresiones de rutas de acceso sencillas en las bases de datos orientadas a objetos y relacionales orientadas a objetos (véase el Apartado 9.5.1).

Una **expresión de ruta** en XPath es una secuencia de pasos de ubicación separados por «/» (en lugar del operador «.» que separa pasos en SQL:1999). El resultado de la expresión de ruta es un conjunto de valores. Por ejemplo, en el documento de la Figura 10.8 la expresión XPath

/banco-2/cliente/name

devolverá estos elementos:

<name>Juncal</name>
<name>Loreto</name>
<name>María</name>

La expresión

/banco-2/cliente/name/text()

devolverá los mismos nombres, pero sin las etiquetas de cierre.

Como en una jerarquía de directorios el signo '/' inicial indica la raíz del documento (nótese que esto es una raíz abstracta «sobre» <banco-2>, que es la marca del documento). Las expresiones de ruta se evalúan de izquierda a derecha. Cuando se evalúa la expresión de ruta el resultado de la ruta en cualquier punto consiste en un conjunto de nodos del documento.

Cuando un nombre de elemento, tal como cliente, aparece antes de la siguiente '/' se refiere a todos los elementos del nombre especificado que son hijos de elementos en el conjunto de elementos actual. Puesto que varios hijos pueden tener el mismo nombre, el número de nodos en el conjunto de nodos puede aumentar o decrecer con cada paso. También se puede acceder a los valores de atributo mediante el uso del símbolo «@». Por ejemplo, /banco-2/cuenta/@número-cuenta devuelve un conjunto con todos los valores de los atributos número-cuenta de los elementos cuenta. De forma predeterminada no se siguen los vínculos IDREF; posteriormente veremos cómo tratar con IDREF.

XPath soporta otras características:

- La selección de predicados puede seguir cualquier paso en una ruta y están contenidos entre corchetes. Por ejemplo

  /banco-2/cuenta[saldo > 400]

  devuelve los elementos cuenta con un valor saldo mayor que 400 mientras que

  /banco-2/cuenta[saldo > 400]/@número-cuenta

  devuelve los números de cuenta de dichas cuentas.

  Se puede comprobar la existencia de un subelemento mediante su listado sin ninguna operación de comparación; por ejemplo si se elimina «>400» de la expresión anterior devolvería los números de cuenta de todas las cuentas que tiene un subelemento saldo, sin considerar su valor.

- XPath proporciona varias funciones que se pueden usar como parte de predicados incluyendo la comprobación de la posición del nodo actual en el orden de los hermanos y contando el número de nodos coincidentes. Por ejemplo, la expresión de ruta

  /banco-2/cuenta/[cliente/count()> 2]

  devuelve las cuentas con más de dos clientes. Se pueden usar las conectivas lógicas and y or en los predicados y la función not(…) se puede usar para la negación.

- La función id(«foo») devuelve el nodo (si existe) con un atributo del tipo ID y cuyo valor sea «foo». La función id se puede incluso aplicar a conjuntos de referencias o incluso a cadenas que contengan referencias múltiples separadas por espacios vacíos, tales como IDREFS.
  Por ejemplo, la ruta

  /banco-2/cuenta/id(@tenedores)

  devuelve todos los clientes referenciados desde el atributo tenedores de los elementos cuenta.

- El operador | permite unir resultados de expresiones. Por ejemplo, si la DTD de banco-2 también contiene elementos para préstamos con atributo prestamista del tipo IDREFS para identificar el prestamista de un préstamo, la expresión

  /banco-2/cuenta/id(@tenedores) | /banco-2/préstamo/id(@prestamista)

  proporciona los clientes con cuentas o préstamos. Sin embargo, el operador | no se puede anidar dentro de otros operadores.

- Una expresión XPath puede saltar varios niveles de nodos mediante el uso de «//». Por ejemplo la expresión /banco-2//name encuentra cualquier elemento name en *cualquier lugar* bajo el elemento /banco-2, sin considerar el elemento en el que está contenido. Este ejemplo ilustra la capacidad de buscar datos requeridos sin un conocimiento completo del esquema.

- Cada paso en la ruta no selecciona los hijos de los nodos del conjunto actual de nodos. En realidad esto es solamente una de las distintas direcciones que puede tomar un paso en la ruta tales como padres, hermanos, ascendientes y descendientes. Aquí se omiten los detalles, pero nótese que «//», descrito anteriormente, es una forma abreviada de especificar «todos los descendientes» mientras que «..» especifica el padre.

10.4.2. XSLT

Una **hoja de estilo** es una representación de las opciones de formato para un documento, normalmente almacenado fuera del documento mismo, por lo que el formato está separado del contenido. Por ejemplo, una hoja de estilo para HTML podría especificar la fuente a usar

en todas la cabeceras y por ello reemplaza un gran número de declaraciones de fuente en la página HTML. **XSL** (**XML *Stylesheet Language***), el lenguaje de hojas de estilo XML, estaba originalmente diseñado para generar HTML a partir de XML y es por ello una extensión lógica de hojas de estilo HTML. El lenguaje incluye un mecanismo de transformación de propósito general, denominado **XSLT** (**XSL *Transformations***, transformaciones XSL), que se puede usar para transformar un documento XML en otro documento XML, o a otros formatos como HTML[1]. Las transformaciones XSLT son bastante potentes y en realidad XSLT puede incluso actuar como un lenguaje de consulta.

Las transformaciones XSLT se expresan como una serie de reglas recursivas, denominadas **plantillas**.

En su forma básica las plantillas permiten la selección de nodos en un árbol XML mediante una expresión XPath. Sin embargo, las plantillas también pueden generar contenido XML nuevo de forma que esa selección y generación de contenido se pueda mezclar de formas naturales y potentes. Aunque XSLT se puede usar como un lenguaje de consulta, su sintaxis y semántica es bastante distinta a la de SQL.

Una plantilla sencilla para XSLT consiste en una parte de coincidencia (match) y una parte de selección (select). Consideremos el siguiente código XSLT.

```
<xsl:template match = «/banco-2/cliente»>
    <xsl:value-of select = «nombre-cliente»/>
</xsl:template>
<xsl:template match = «*»/>
```

La instrucción xsl:template match contiene una expresión XPath que selecciona uno o más nodos. La primera plantilla busca coincidencias de elementos cliente que aparecen como hijos del elemento raíz banco-2. La instrucción xsl:value-of encerrada en la instrucción de coincidencia devuelve valores de los nodos en el resultado de la expresión XPath. La primera plantilla devuelve el valor del subelemento nombre-cliente; nótese que el valor no contiene la marca element.

Nótese que la segunda plantilla coincide con todos los nodos. Esto se requiere porque el comportamiento predeterminado de XSLT sobre los elementos del documento de entrada que no coinciden con ninguna plantilla es copiar su contenido textual en el documento de salida y aplicar recursivamente las plantillas a sus subelementos.

Cualquier texto o etiqueta de la hoja de estilo XSLT que no esté en el espacio de nombres xsl se copia a la salida sin cambios. La Figura 10.10 muestra cómo usar esta característica para hacer que cada nombre de cliente del ejemplo aparezca como un subelemento del elemento «<cliente>» mediante la ubicación de la instrucción xsl:value-of entre <cliente> y </cliente>.

[1] El estándar XSL ahora consiste en XSLT y un estándar para especificar las características del formato tales como fuentes, márgenes de página y tablas. El formato no es relevante desde una perspectiva de las bases de datos, por lo que no se tratará aquí.

```
<xsl:template match = «/banco-2/cliente»>
    <cliente>
    <xsl:value-of select = «nombre-cliente»/>
    </cliente>
</xsl:template>
<xsl:template match = «*»/>
```

FIGURA 10.10. Uso de XSLT para convertir los resultados en nuevos elementos XML.

La creación de un atributo, como id-Cliente, en el elemento Cliente generado, es más complicado y requiere el uso de xsl:attribute. Véase un manual de XSLT para más detalles.

La **recursividad estructural** es una parte clave de XSLT. Hay que recordar que los elementos y subelementos naturalmente forman una estructura en árbol. La idea de la recursividad estructural es la siguiente: cuando una plantilla coincide con un elemento en la estructura de árbol XSLT puede usar la recursividad estructural para aplicar las reglas de la plantilla recursivamente a los subárboles en lugar de simplemente devolver un valor. Aplica las reglas recursivamente mediante la directiva xsl:apply-templates, que aparece dentro de otras plantillas.

Por ejemplo, los resultados de nuestra consulta anterior se pueden ubicar en un elemento <clientes> mediante la adición de una regla usando xsl:apply-templates, como en la Figura 10.11. La nueva regla coincide con la etiqueta externa «banco» y construye un documento resultado mediante la aplicación de otras plantillas a los subárboles que aparecen en el elemento banco, pero envolviendo los resultados en el elemento <clientes> </clientes> dado. Sin la recursividad forzada por la cláusula <xsl:apply-templates/> la plantilla devolvería <clientes> </clientes> y entonces aplicaría otras plantillas a los subelementos.

En realidad, la recursividad estructural es crítica para construir documentos XML bien formados, puesto que los documentos XML deben tener un único elemento de nivel superior que contenga el resto de elementos del documento.

XSLT proporciona una característica denominada **key** (clave), que permite la búsqueda de elementos mediante el uso de valores de subelementos o atributos; los objetivos son similares a los de la función id() en XPath, pero permite usar atributos distintos a los atri-

```
<xsl:template match = «/banco»>
    <clientes>
    <xsl:apply-templates/>
    </clientes>
</xsl:template>
<xsl:template match = «/cliente»>
    <cliente>
    <xsl:value-of select = «nombre-cliente»/>
    </cliente>
</xsl:template>
<xsl:template match = «*»/>
```

FIGURA 10.11. Aplicación recursiva de reglas.

butos ID. Las claves se definen mediante una directiva xsl:key la cual tiene tres partes, por ejemplo:

```
<xsl:key name = «numcuenta» match =
  «cuenta» use = «número-cuenta»/>
```

El atributo name se usa para distinguir claves distintas. El atributo match especifica los nodos a los que se aplica la clave. Finalmente, el atributo use especifica la expresión a usar como el valor de la clave. Nótese que la expresión no tiene que ser única para un elemento; esto es, más de un elemento puede tener el mismo valor de expresión. En el ejemplo la clave denominada numcuenta especifica que el subelemento número-cuenta de cuenta se debería usar como una clave para esa cuenta.

Las claves se pueden usar en plantillas como parte de cualquier patrón mediante la función key. Esta función toma el nombre de la clave y un valor y devuelve el conjunto de nodos que coinciden con ese valor. Por ello, el nodo XML para la cuenta «C-401» se puede referenciar como key(«numcuenta», «C-401»).

Las claves se pueden usar para implementar algunos tipos de reuniones, como en la Figura 10.12. El código de la figura se puede aplicar a datos XML en el formato de la Figura 10.1. Aquí la función key reúne los elementos impositor con los elementos coincidentes cliente y cuenta. El resultado de la consulta consiste en pares de elementos cliente y cuenta encerrados en elementos cuenta-cliente.

XSLT permite ordenar los nodos. Un ejemplo sencillo muestra cómo se usaría xsl:sort en la hoja de estilo para devolver los elementos cliente ordenados por el nombre:

```
<xsl:template match = «/banco»>
    <xsl:apply-templates select = «cliente»>
    <xsl:sort select = «nombre-cliente»/>
    </xsl:apply-templates>
</xsl:template>
<xsl:template match = «cliente»>
    <cliente>
        <xsl:value-of select = «nombre-cliente»/>
        <xsl:value-of select = «calle-cliente»/>
        <xsl:value-of select = «ciudad-cliente»/>
    </cliente>
</xsl:template>
<xsl:template match = «*»/>
```

Aquí xsl:apply-template tiene un atributo select que lo restringe para que sólo se aplique a los subelementos cliente. La directiva xsl:sort en el elemento xsl:apply-template hace que los nodos se ordenen antes de ser procesados por el siguiente conjunto de plantillas. Existen opciones para permitir la ordenación sobre varios subelementos/attributes, por valor numérico y en orden descendente.

10.4.3. XQuery

El consorcio W3C (World Wide Web Consortium) está desarrollando XQuery, un lenguaje de consulta de XML. Nuestra discusión aquí está basada en un bosquejo del lenguaje estándar, por lo que el estándar final puede diferir; sin embargo se espera que las principales características no cambien sustancialmente. El lenguaje XQuery se deriva de un lenguaje de consulta XML denominado Quilt; la mayor parte de las características de XQuery que se analizan aquí son parte de Quilt. Quilt por sí mismo incluye características de lenguajes anteriores, tales como XPath, discutido en el Apartado10.4., y otros dos lenguajes de consulta XML: XQL y XML-QL.

A diferencia de XSLT, XQuery no representa consultas en XML. En su lugar se parece más a consultas SQL y se organizan en expresiones «FLWR» (pronunciado «flower», «flor» en inglés) que comprende cuatro secciones: **for**, **let**, **where** y **return**. La sección **for** proporciona una serie de variables que cuyos valores son los resultados de expresiones XPath. Cuando se especifica más de una variable, los resultados incluyen el producto cartesiano de los valores posibles que las variables pueden tomar, haciendo la cláusula **for** similar en espíritu a la cláusula **from** de la consulta SQL. La cláusula **let** simplemente permite que se asignen expresiones complicadas a los nombres de las variables por simplicidad de representación. La sección **where**, como la cláusula SQL **where**, ejecuta comprobaciones adicionales sobre las tuplas reunidas de la sección **for**. Finalmente la sección **return** permite la construcción de resultados en XML.

Una expresión FLWR sencilla que devuelve los números de cuentas para las cuentas corrientes está basada en el documento XML de la Figura 10.8, que usa ID e IDREFS:

```
<xsl:key name = «numcuenta» match = «cuenta» use = «número-cuenta»/>
<xsl:key name = «numcliente» match = «cliente» use = «nombre-cliente»/>
<xsl:template match = «impositor»>
    <cuenta-cliente>
    <xsl:value-of select = key(«numcliente», «nombre-cliente»)/>
    <xsl:value-of select = key(«numcuenta», «número-cuenta»)/>
    </cuenta-cliente>
</xsl:template>
<xsl:template match = «*»/>
```

FIGURA 10.12. Combinaciones en XSLT.

```
for $x in /banco-2/cuenta
let $numcuenta : = $x/@número-cuenta
where $x/saldo > 400
return <número-cuenta> $numcuenta </número-
    cuenta>
```

Puesto que esta consulta es sencilla, la cláusula **let** no es esencial y la variable $numcuenta en la cláusula **return** se podría remplazar con $x/@número-cuenta. Nótese además que, puesto que la cláusula **for** usa expresiones XPath, se pueden producir selecciones en la expresión XPath. Por ello, una consulta equivalente puede tener solamente cláusulas **for** y **return**:

```
for $x in /banco-2/cuenta[saldo > 400]
return <número-cuenta> $x/@número-cuenta
    </número-cuenta>
```

Sin embargo, la cláusula **let** simplifica las consultas complejas.

Las expresiones de ruta en XQuery pueden devolver un multiconjunto con nodos repetidos. La función distinct aplicada a un multiconjunto devuelve un conjunto sin duplicación. La función distinct se puede usar incluso con una cláusula **for**. XQuery también proporciona funciones de agregado tales como sum y count que se pueden aplicar a colecciones tales como conjuntos y multiconjuntos. Aunque XQuery no proporciona una constructora **group by**, las consultas de agregado se pueden escribir mediante el uso de constructoras FLWR anidadas en lugar de agrupamientos; dejamos los detalles como ejercicio. Nótese también que las variables asignadas por las cláusulas **let** pueden tener valores de conjunto o multiconjunto si la expresión de ruta en el lado derecho devuelve un conjunto o un multiconjunto respectivamente.

Las reuniones se especifican en XQuery de forma parecida a SQL. La reunión de elementos impositor, cuenta y cliente en la Figura 10.1, que se escribió en XSLT en el Apartado 10.4.2, se puede escribir en XQuery de la siguiente forma

```
for $a in /banco/cuenta,
    $c in /banco/cliente,
    $i in /banco/impositor
where $a/número-cuenta = $i/número-cuenta
    and $c/nombre-cliente = $i/nombre-cliente
return <cuenta-cliente> $c $a </cuenta-cliente>
```

La misma consulta se puede expresar con las selecciones especificadas como selecciones XPath:

```
for $a in /banco/cuenta,
    $c in /banco/cliente,
    $i in /banco/impositor[número-cuenta =
        $a/número-cuenta
    and nombre-cliente = $c/nombre-cliente]
return <cuenta-cliente> $c $a</cuenta-cliente>
```

Las expresiones XQuery FLWR se pueden anidar en la cláusula **return** con el fin de generar anidamientos de elementos que no aparecen en el documento origen. Esta característica es similar a las subconsultas anidadas en la cláusula **from** de las consultas SQL del Apartado 9.5.3.

Por ejemplo, la estructura XML mostrada en la Figura 10.3, con los elementos de la cuenta anidados en elementos cliente, se puede generar a partir de la estructura en la Figura 10.1 mediante esta consulta:

```
<banco-1>
    for $c in /banco/cliente
    return
        <cliente>
            $c/*
            for $i in /banco/impositor[nombre-cliente =
                $c/nombre-cliente],
                $a in /banco/cuenta[número-cuenta =
                $i/número-cuenta]
            return $a
        </cliente>
</banco-1>
```

La consulta también introduce la sintaxis $c/*, que se refiere a todos los hijos del nodo, que está ligada a la variable $c. De forma similar, $c/text() proporciona el contenido textual de un elemento, sin las etiquetas.

Las expresiones de ruta en XQuery están basadas en expresiones de ruta en XPath, pero XQuery proporciona algunas extensiones (que se pueden finalmente agregar al mismo XPath). Una de las extensiones de sintaxis útiles es el operador ->, que se puede usar para desreferenciar IDREFs, al igual que la función id(). Se puede aplicar el operador sobre un valor del tipo IDREFS para obtener un conjunto de elementos. Se puede usar, por ejemplo, para encontrar todas las cuentas asociadas con un cliente, con la representación ID/IDREFS de la información bancaria. Dejamos los detalles al lector.

En XQuery los resultados se pueden ordenar si se incluye una cláusula **sortby** al final de cualquier expresión; la cláusula especifica cómo se han de ordenar las instancias de esa expresión. Por ejemplo, esta consulta tiene como salida todos los elementos cliente ordenados por el subelemento name:

```
for $c in /banco/cliente,
return <cliente> $c/* </cliente> sortby(name)
```

Para ordenar de forma decreciente podemos usar **sortby**(name **descending**).

La ordenación se puede realizar en varios niveles de anidamiento. Por ejemplo se puede obtener una representación anidada de la información bancaria ordenada según el nombre del cliente, con las cuentas de cada cliente ordenadas según el número de cuenta, según sigue:

```
<banco-1>
    for $c in /banco/cliente
    return
        <cliente>
            $c/*
            for $d in /banco/impositor[nombre-cliente =
                $c/nombre-cliente],
                $a in /banco/cuenta[número-cuenta =
                    $d/número-cuenta]
            return <cuenta> $a/* </cuenta> sortby
                (número-cuenta)
        </cliente> sortby(nombre-cliente)
</banco-1>
```

XQuery proporciona una serie de funciones incorporadas y soporta funciones definidas por el usuario. Por ejemplo, la función incorporada document(name) devuelve la raíz de un documento con nombre; la raíz se puede usar entonces en una expresión de ruta para acceder al contenido del documento. Los usuarios pueden definir funciones tal como se ilustra con esta función, que devuelve una lista de todos los saldos de un cliente con un nombre especificado:

```
function saldos(xsd:string $c) returns list(xsd:
        numeric) {
    for $i in /banco/impositor[nombre-cliente = $c],
        $a in /banco/cuenta[número-cuenta =
        $i/número-cuenta]
    return $a/saldo
}
```

XQuery usa el sistema de tipos de XMLSchema. XQuery también proporciona funciones para realizar conversiones entre tipos. Por ejemplo, number(x) convierte una cadena a un número. XQuery ofrece una gran variedad de otras características, tales como cláusulas if-then-else, las cuales se pueden usar con cláusulas **return**, y la cuantificación existencial y universal, que se pueden usar en predicados en cláusulas **where**. Por ejemplo, la cuantificación existencial se puede expresar mediante el uso de **some** $e **in** path **satisfies** P donde path es una expresión de ruta y P es un predicado que puede usar $e. La cuantificación universal se puede expresar mediante el uso de **every** en lugar de **some**.

10.5. LA INTERFAZ DE PROGRAMACIÓN DE APLICACIONES

Debido a la gran aceptación de XML como una representación de datos y formato de intercambio hay gran cantidad de herramientas de software disponibles para la manipulación de datos XML. En realidad hay dos modelos estándar para la manipulación mediante programación de XML, cada uno disponible para su uso con una gran cantidad de lenguajes de programación populares.

Una de las API estándar para la manipulación XML es el *modelo de objetos documento* (Document Object Model, DOM), que trata el contenido XML como un árbol, con cada elemento representado por un nodo, denominado DOMNode. Los programas pueden acceder a partes del documento mediante navegación comenzando con la raíz.

Hay disponibles bibliotecas DOM para los lenguajes de programación más comunes y están incluso presentes en los exploradores Web, donde se pueden usar para manipular el documento mostrado al usuario. Aquí se explican algunas de las interfaces y métodos de la API DOM de Java, para mostrar cómo puede ser un DOM. La API DOM de Java proporciona una interfaz denominada Node e interfaces Element y Attribute las cuales heredan de la interfaz Node. La interfaz Node proporciona métodos tales como getParentNode(), getFirstChild() y getNextSibling() para navegar por el árbol DOM comenzando por el nodo raíz. Se puede acceder a los subelementos de un elemento mediante el nombre getElementsByTagName(name) que devuelve una lista de todos los elementos hijo con un nombre de etiqueta especificado; se puede acceder a los miembros individuales de la lista mediante el método hijo, que devuelve el *i*-ésimo elemento en la lista. Se puede acceder a los valores de atributo de un elemento mediante el nombre, usando el método getAttribute(name). El valor de texto de un elemento se modela como un nodo Text, que es un hijo del nodo elemento; un nodo elemento sin subelemento tiene solamente un nodo hijo. El método getData() del nodo Text devuelve el contenido de texto. DOM también proporciona una serie de funciones para actualizar el documento mediante la adición y el borrado de hijos elemento y atributo, el establecimiento de valores de nodos, etc.

Se requieren muchos más detalles para escribir un programa DOM real; véanse las notas bibliográficas para obtener referencias con más información.

DOM se puede usar para acceder a los datos XML almacenados en las bases de datos y se puede construir una base de datos XML mediante el uso de DOM como su interfaz principal para acceder y modificar los datos. Sin embargo, la interfaz DOM no soporta ninguna forma de consulta declarativa.

La segunda interfaz de programación que discutiremos, *API simple para XML* (Simple API for XML, SAX) es un modelo de *eventos* diseñado para proporcionar una interfaz común entre analizadores y aplicaciones. Esta API está construida bajo la noción de

manejadores de eventos, que consisten en funciones especificadas por el usuario asociadas con eventos de análisis. Los eventos de análisis corresponden con el reconocimiento de partes de un documento; por ejemplo, se genera un evento cuando se encuentra la etiqueta de inicio para un elemento y se genera otro evento cuando se encuentra la etiqueta de finalización. Las piezas de un documento siempre se encuentran en orden desde el inicio al final. SAX no es adecuado para aplicaciones de bases de datos.

10.6. ALMACENAMIENTO DE DATOS XML

Muchas aplicaciones requieren el almacenamiento de datos XML. Una forma de almacenar datos XML es convertirlos a una representación relacional y almacenarlos en una base de datos relacional. Hay varias alternativas para almacenar datos XML, las cuales se muestran brevemente aquí.

10.6.1. Bases de datos relacionales

Puesto que las bases de datos relacionales se usan ampliamente en aplicaciones existentes es una gran ventaja almacenar datos XML en bases de datos relacionales de forma que se pueda acceder a los datos desde aplicaciones existentes.

La conversión de datos XML a una forma relacional es normalmente muy sencilla si los datos se han generado en un principio desde un esquema relacional y XML se usó simplemente como un formato de intercambio de datos para datos relacionales. Sin embargo, hay muchas aplicaciones donde los datos XML no se han generado desde un esquema relacional y la traducción de los datos a una forma relacional de almacenamiento puede no ser tan sencilla. En particular los elementos anidados y los elementos que se repiten (correspondientes a atributos con valores de conjunto) complican el almacenamiento de los datos XML en un formato relacional. Hay disponibles diversas alternativas.

- **Almacenamiento como cadena.** Una forma sencilla de almacenar los datos XML en una base de datos relacional es almacenar cada elemento hijo del elemento de mayor nivel como una cadena en una tupla separada de la base de datos. Por ejemplo, los datos XML de la Figura 10.1 se podrían almacenar como un conjunto de tuplas en una relación *elementos(datos)*, con el atributo *datos* de cada tupla almacenando un elemento XML (cuenta, cliente o impositor) en forma de cadena.

  Aunque esta representación es fácil de usar, el sistema de la base de datos no conoce el esquema de los elementos almacenados. Como resultado no es posible consultar los datos directamente. En realidad no es siquiera posible implementar selecciones sencillas tales como buscar todos los elementos cuenta o encontrar el elemento cuenta cuyo número de cuenta sea C-401, sin explorar todas las tuplas de la relación y examinar los contenidos de la cadena almacenada en la tupla.

  Una solución parcial a este problema es almacenar distintos tipos de elementos en relaciones diferentes y también almacenar los valores de algunos elementos críticos como atributos de la relación que permite la indexación. Así, en nuestro ejemplo, las relaciones serían *elementos-cuenta*, *elementos-cliente* y *elementos-impositor*, cada una con un atributo *datos*. Cada relación puede tener atributos extra para almacenar los valores de algunos subelementos tales como *número-cuenta* o *nombre-cliente*. Por ello se puede responder eficientemente con esta representación a una consulta que requiera los elementos cuenta con un número de cuenta especificado. Tal enfoque depende del tipo de información sobre los datos XML, tales como la DTD de los datos.

  Algunos sistemas de bases de datos, tales como Oracle 9, soportan **índices de función**, que pueden ayudar a evitar la duplicación de atributos entre la cadena XML y los atributos de la relación. A diferencia de los índices normales, que se construyen sobre los valores de atributos, los índices de función se pueden construir sobre el resultado de aplicar funciones definidas por el usuario a las tuplas. Por ejemplo, se puede construir un índice de función sobre una función definida por el usuario que devuelve el valor del subelemento número-cuenta de la cadena XML en una tupla. El índice se puede entonces usar de la misma forma que un índice sobre un atributo *número-cuenta*.

  Estos enfoques tienen el inconveniente de que una gran parte de la información XML se almacena en cadenas. Es posible almacenar toda la información en relaciones según alguna de las siguientes formas que se examinan a continuación.

- **Representación en árbol.** Los datos XML arbitrarios se pueden modelar como un árbol y almacenar mediante el uso de un par de relaciones:

  nodos(*id*, *tipo*, *etiqueta*, *valor*)
  hijo(*id-hijo*, *id-padre*)

  A cada elemento y atributo de los datos XML se le proporciona un identificador único. Una tupla insertada en la relación *nodos* para cada elemento y atributo con su identificador (*id*), su tipo (*atributo o elemento*), el nombre del elemento o atributo (*etiqueta*) y el valor textual del elemento o atribu-

to (*valor*). La relación *hijo* se usa para guardar el elemento padre de cada elemento y atributo. Si la información de orden de los elementos y atributos se debe preservar, se puede agregar un atributo extra *posición* a la relación *hijo* para indicar la posición relativa del hijo entre los hijos del padre. Como ejercicio se pueden representar los datos XML de la Figura 10.1 mediante el uso de esta técnica.

Esta representación tiene la ventaja de que toda la información XML se puede representar directamente de forma relacional y se pueden trasladar muchas consultas XML a consultas relacionales y ejecutar dentro del sistema de la base de datos. Sin embargo, tiene el inconveniente de que cada elemento se divide en muchas piezas y se requieren un gran número de combinaciones para reensamblar los elementos.

- **Asignación a relaciones.** Según este enfoque los elementos XML cuyo esquema es conocido se asignan a relaciones y atributos. Los elementos cuyo esquema es desconocido se almacenan como cadenas o como una representación en árbol.

  Se crea una relación para cada tipo de elemento cuyo esquema sea conocido. Todos los atributos de estos elementos se almacenan como atributos de la relación. Todos los subelementos que suceden más de una vez dentro de estos elementos (según se especifica en DTD) también se pueden representar como atributos de la relación; si el subelemento puede contener solamente texto, el atributo almacena el valor de texto. En otro caso, la relación correspondiente al subelemento almacena los contenidos del subelemento junto con un identificador para el tipo padre y el atributo almacena el identificador del subelemento. Si el subelemento tiene más subelementos anidados se aplica el mismo procedimiento al subelemento.

  Si un subelemento puede aparecer varias veces en un elemento, el enfoque de asignación a relaciones almacena el contenido de los subelementos en la relación correspondiente al subelemento. Proporciona identificadores únicos tanto para el padre como para el subelemento y crea una relación separada, similar a la relación *hijo* vista en la representación en árbol anterior para identificar el subelemento que aparece bajo cada padre. Nótese que cuando se aplica este enfoque a la DTD de los datos en la Figura 10.1 se vuelve al esquema relacional original que se ha usado en capítulos anteriores. Las notas bibliográficas proporcionan referencias a tales enfoques híbridos.

10.6.2. Almacenamientos de datos no relacionales

Hay varias alternativas para almacenar datos XML en sistemas de almacenamientos de datos no relacionales:

- **Almacenamiento en archivos planos.** Puesto que XML es principalmente un formato de archivo, un mecanismo de almacenamiento natural es simplemente un archivo plano. Este enfoque tiene muchos de los inconvenientes mostrados en el Capítulo 1 sobre el uso de sistemas de archivos como base para las aplicaciones de bases de datos. En particular hay una carencia de aislamiento de datos, comprobaciones de integridad, atomicidad, acceso concurrente y seguridad. Sin embargo, la amplia disponibilidad de herramientas XML que funcionan sobre archivos de datos hace relativamente sencillo el acceso y consulta de datos XML almacenados en archivos. Por ello, este formato de almacenamiento puede ser suficiente para algunas aplicaciones.
- **Almacenamiento en una base de datos XML.** Las bases de datos XML son bases de datos que usan XML como su modelo de datos básico. Las bases de datos XML antiguas implementaban el modelo de objetos documento sobre una base de datos orientada a objetos basada en C++. Esto permite reusar gran parte de la infraestructura de bases de datos orientada a objetos mientras se usa una interfaz XML estándar. La adición de un lenguaje de consulta XML proporciona consultas declarativas. También es posible construir bases de datos XML como una capa en la parte superior de las bases de datos relacionales.

10.7. APLICACIONES XML

Un objetivo central en el diseño para XML es hacer más sencillo comunicar la información en Web y entre aplicaciones permitiendo que la semántica de los datos se describa con los mismos datos. Por ello, aunque la gran cantidad de datos XML y su uso en aplicaciones de negocios indudablemente requerirán y se beneficiarán de las tecnologías de bases de datos XML, es principalmente un medio de comunicación. Dos aplicaciones XML para comunicación (intercambio de datos y mediación de recursos de información Web) ilustran cómo XML logra su objetivo de soportar el intercambio de datos y demuestran cómo la tecnología e interacción de las bases de datos son la clave en dar soporte a aplicaciones basadas en el intercambio.

10.7.1. Intercambio de datos

Se están desarrollando estándares para la representación XML de los datos de una gran variedad de aplicaciones especializadas que van desde aplicaciones de

negocios tales como banca y transportes a aplicaciones científicas tales como química y biología molecular. Veamos algunos ejemplos:

- La industria química necesita información sobre los compuestos químicos tales como su estructura molecular y una serie de propiedades importantes tales como los puntos de fusión y ebullición, valores caloríficos, solubilidad en distintos solventes y cosas así. *ChemML* es un estándar para representar dicha información.
- En el transporte, los transportes de mercancías y los agentes de aduana e inspectores necesitan los registros de los envíos que contengan información detallada sobre los bienes que están siendo transportados, por quién y desde dónde se han enviado, a quién y a dónde se envían, el valor monetario de los bienes y cosas así.
- Un mercado en línea en que los negocios pueden vender y comprar bienes (el llamado mercado B2B [business-to-business, de negocio a negocio]) requiere información tal como los catálogos de producto, incluyendo descripciones detalladas de los productos e información de los precios, inventarios de los productos, ofertas a comprar y cuotas para una venta propuesta.

El uso de esquemas relacionales normalizados para modelar requisitos de datos tan complejos genera un gran número de relaciones que son complicadas de gestionar por los usuarios. Las relaciones frecuentemente tienen un gran número de atributos; la representación explícita de nombres de atributos y elementos con sus valores en XML ayuda a evitar confusiones entre los atributos. Las representaciones de elementos anidados ayudan a reducir el número de relaciones que se deben representar, así como el número de combinaciones requeridas para obtener la información, con el posible coste de redundancia. Así, en nuestro ejemplo bancario el listado de clientes con elementos cuenta anidados en elementos cuenta, como en la Figura 10.3, resulta en un formato que es más natural para algunas aplicaciones, en particular para su legibilidad, que lo que es la representación normalizada de la Figura 10.1.

Cuando se usa XML para intercambiar los datos entre aplicaciones de negocio los datos se originan muy frecuentemente en bases de datos relacionales. Los datos en las bases de datos relacionales deben ser *publicadas*, esto es, convertidos a formato XML para su exportación a otras aplicaciones. Los datos de entrada deben ser *despiezados*, es decir, convertida desde XML a un formato normalizado de relación y almacenado en una base de datos relacional. Aunque el código de la aplicación pueda ejecutar las operaciones de publicación y despiece, las operaciones son tan comunes que la conversión se debería realizar de forma automática, sin escribir ningún código en la aplicación, siempre que sea posible. Por ello, los fabricantes de bases de datos están trabajando para dar *capacidades XML* a sus productos de bases de datos.

Una base de datos con capacidades XML permite una correspondencia automática de su modelo relacional interno (relacional, relacional orientado a objetos u orientado a objetos) con XML. Estas correspondencias pueden ser sencillas o complejas. Una correspondencia sencilla podría asignar un elemento a cada fila de una tabla y hacer de cada columna en esa fila bien un atributo o bien un subelemento del elemento de la fila. Dicha correspondencia es sencilla de generar automáticamente. Una correspondencia más complicada necesitaría que se crearan estructuras anidadas. Las extensiones de SQL con consultas anidadas en la cláusula **select** se han desarrollado para permitir una creación fácil de salidas XML anidadas. Algunos productos de bases de datos permiten a las consultas XML acceder a los datos relacionales tratando la forma XML de los datos relacionales como un documento XML *virtual*.

10.7.1.1. Mediación de datos

La compra comparada es un ejemplo de aplicación de mediación de datos en la que los datos sobre elementos, inventario, precio y costes de envío se extraen de una serie de sitios Web que ofrecen un elemento en particular de venta. La información agregada resultante es significativamente más valiosa que la información individual ofrecida por un único sitio.

Un gestor financiero personal es una aplicación similar en el contexto de la banca. Consideremos un consumidor con una gran cantidad de cuentas a gestionar, tales como cuentas bancarias, cuentas de ahorro y cuentas de jubilación. Supongamos que estas cuentas pueden estar en distintas instituciones. Es un reto importante proporcionar una gestión centralizada de todas las cuentas de un cliente. La mediación basada en XML soluciona el problema extrayendo una representación XML de la información de la cuenta desde los sitios Web respectivos de las instituciones financieras donde están las cuentas individuales. Esta información se puede extraer fácilmente si la institución la exporta a un formato XML estándar, e indudablemente algunas lo harán. Para aquellas que no lo hacen se usa un software *envolvente* para generar datos XML a partir de las páginas Web HTML devueltas por el sitio Web. Las aplicaciones envolventes necesitan un mantenimiento constante, puesto que dependen de los detalles de formato de las páginas Web, que cambian constantemente. No obstante, el valor proporcionado por la mediación frecuentemente justifica el esfuerzo requerido para desarrollar y mantener las aplicaciones envolventes.

Una vez que las herramientas básicas están disponibles para extraer la información de cada fuente, se usa una aplicación mediadora para combinar la información extraída bajo un único esquema. Esto puede requerir más transformación de los datos XML de cada sitio,

puesto que los distintos sitios pueden estructurar la misma información de una forma diferente. Por ejemplo, uno de los bancos puede exportar información en el formato de la Figura 10.1 aunque otros pueden usar la formato anidado de la Figura 10.3. También pueden usar nombres diferentes para la misma información (por ejemplo num-cuenta e id-cuenta), o pueden incluso usar el mismo nombre para información distinta. El mediador debe decidir sobre un único esquema que representa toda la información requerida, y debe proporcionar código para transformar los datos entre diferentes representaciones. Dichos temas se discutirán con mayor detalle en el Apartado 19.8 en el contexto de bases de datos distribuidas. Los lenguajes de consulta XML tales como XSLT y XQuery juegan un papel importante en la tarea de transformación entre distintas representaciones XML.

10.8. RESUMEN

- Al igual que el lenguaje de marcas de hipertexto, HTML (*Hyper-Text Markup Language*), en que está basado Web, el lenguaje de marcas extensible, XML (*Extensible Markup Language*), es un descendiente del lenguaje estándar generalizado de marcas (SGML, *Standard Generalized Markup Language*). XML tuvo la intención original de proporcionar marcas funcionales para documentos Web, pero se ha convertido ahora en un formato de datos estándar para el intercambio entre aplicaciones.
- Los documentos XML contienen elementos con etiquetas de inicio y finalización correspondientes que indican el comienzo y finalización de un elemento. Los elementos puede tener subelementos anidados a ellos, a cualquier nivel de anidamiento. Los elementos pueden también tener atributos. La elección entre representar información como atributos y subelementos es frecuentemente arbitraria en el contexto de la representación de datos.
- Los elementos pueden tener un atributo del tipo ID que almacene un identificador único para el elemento. Los elementos también pueden almacenar referencias a otros elementos mediante el uso de atributos del tipo IDREF. Los atributos del tipo IDREFS pueden almacenar una lista de referencias.
- Los documentos pueden opcionalmente tener su esquema especificado mediante una definición de tipos de documento (DTD, *Document Type Declaration*). La DTD de un documento especifica los elementos que pueden aparecer, cómo se pueden anidar y los atributos que puede tener cada elemento.
- Aunque los DTDs se usan ampliamente, tienen varias limitaciones. Por ejemplo, no proporcionan un sistema de tipos. XMLSchema es un nuevo estándar para especificar el esquema de un documento. Aunque proporciona mayor potencia expresiva, incluyendo un potente sistema de tipos, también es más complicado.
- Los datos XML se pueden representar como estructuras en árbol, como nodos correspondientes a los elementos y atributos. El anidamiento de elementos se refleja mediante la estructura padre-hijo de la representación en árbol.
- Las expresiones de ruta se pueden usar para recorrer la estructura de árbol XML y así localizar los datos requeridos. XPath es un lenguaje estándar para las expresiones de rutas de acceso y permite especificar los elementos requeridos mediante una ruta parecida a un sistema de archivos y además permite la selección y otras características. XPath también forma parte de otros lenguajes de consulta XML.
- El lenguaje XSLT se diseñó originalmente como el lenguaje de transformación para una aplicación de hojas de estilo, en otras palabras, para aplicar información de formato a documentos XML. Sin embargo, XSLT ofrece características bastante potentes de consulta y transformación y está ampliamente disponible, por lo que se usa para consultar datos XML.
- Los programas XSLT contienen una serie de plantillas, cada una con una parte **match** y una parte **select**. Cada elemento en la entrada de datos XML se comprueba con las plantillas disponibles y se aplica al elemento la parte de la selección de la primera plantilla coincidente.

  Las plantillas se pueden aplicar recursivamente desde el cuerpo de otra plantilla, un procedimiento conocido como recursividad estructural. XSLT soporta claves que se pueden usar para implementar algunos tipos de reuniones. También soporta la ordenación y otras características de consulta.
- El lenguaje XQuery, que se está convirtiendo actualmente en un estándar, está basado en el lenguaje de consulta Quilt. El lenguaje XQuery es similar a SQL, con cláusulas **for**, **let**, **where** y **return**.

  Sin embargo, soporta muchas extensiones para tratar con la naturaleza en árbol de XML y para permitir la transformación de documentos XML en otros documentos con una estructura significativamente diferente.
- Los datos XML se pueden almacenar de varias formas distintas. Por ejemplo, los datos XML se pueden almacenar como cadenas en una base de datos relacional. De forma alternativa las relaciones pueden representar los datos XML como árboles. Como otra alternativa, los datos XML se pueden hacer corres-

ponder con relaciones de la misma forma que se hacen corresponder los esquemas E-R con esquemas relacionales.

Los datos XML también se pueden almacenar en sistemas de archivos o en bases de datos XML, las cuales usan XML como su representación interna.

- La capacidad de transformar documentos en lenguajes como XSLT y XQuery es clave para el uso de XML en aplicaciones de mediación tales como intercambios electrónicos de negocios y la extracción y combinación de datos Web para su uso por un gestor financiero personal o compra comparada.

TÉRMINOS DE REPASO

- Almacenamiento de datos XML
  - En bases de datos relacionales
    - Almacenamiento como cadena
    - Representación en árbol
    - Asignación a relaciones
  - En almacenamientos de datos no relacionales
    - Archivos
    - Bases de datos XML
- API simple para XML (Simple API for XML, SAX)
- API XML
- Aplicaciones XML
  - Intercambio de datos
    - Publicación y despiece
  - Mediación de datos
    - Software envolvente
- Atributos
- Autodocumentado
- Base de datos con capacidades XML
- Consulta y transformación
- Definición del esquema
  - Definición de tipos de documento (Document Type Definition, DTD)
  - XMLSchema
- Elemento
- Elemento raíz
- Elementos anidados
- Espacio de nombres
- Espacio de nombres predeterminado
- Expresiones de ruta
- Hoja de estilo
- ID
- IDREF e IDREFS
- Lenguaje de marcas
- Lenguaje de marcas de hipertexto (Hyper-Text Markup Language, HTML)
- Lenguaje de marcas extensible (Extensible Markup Language, XML)
- Lenguaje estándar generalizado de marcas (Standard Generalized Markup Language, SGML)
- Marcas
- Modelo de árbol de datos XML
- Modelo de objetos documento (Document Object Model, DOM)
- Nodos
- Transformaciones XSL (XSL Transformations, XSLT)
  - Plantillas
    - match (coincidencia)
    - select (selección)
  - Recursividad estructural
  - Claves
  - Ordenación
- XPath
- XQuery
  - Expresiones FLWR
    - **for**
    - **let**
    - **where**
    - **return**
  - Reuniones
  - Expresión FLWR anidada
  - Ordenación
- XSL (XML Stylesheet Language), lenguaje de hojas de estilo XML

EJERCICIOS

10.1 Dese una representación alternativa de la información bancaria que contenga los mismos datos que en la Figura 10.1, pero usando atributos en lugar de subelementos. Dese también la DTD para esta representación.

10.2 Demuéstrese, proporcionando una DTD, cómo representar la relación anidada *libros* del Apartado 9.1 mediante el uso de XML.

10.3 Dese la DTD para una representación XML del siguiente esquema relacional anidado

Emp = (*nombree*, *ConjuntoHijos* **setof**(*Hijos*), *ConjuntoMaterias* **setof**(*Materias*))
Hijos = (*nombre*, *Cumpleaños*)
Cumpleaños = (*día*, *mes*, *año*)
Materias = (*tipo*, *ConjuntoExámenes* **setof**(*Exámenes*))
Exámenes = (*año*, *ciudad*)

10.4 Escríbanse las siguientes consultas en XQuery, asumiendo la DTD del Ejercicio 10.3.

a. Encontrar los nombres de todos los empleados que tienen un hijo cuyo cumpleaños cae en marzo.

b. Encontrar aquellos empleados que se examinaron del tipo de materia «mecanografía» en la ciudad «Madrid».

c. Listar todos los tipos de materias en *Emp*.

10.5 Escríbanse las consultas en XSLT y XPath sobre la DTD del Ejercicio 10.3 para listar todos los tipos de materia en *Emp*.

10.6 Escríbase una consulta en XQuery en la representación XML de la Figura 10.1 para encontrar el saldo total en cada sucursal a partir de todas las cuentas. (Consejo: se puede usar una consulta anidada para conseguir el efecto de **group by** de SQL).

10.7 Escríbase una consulta en XQuery en la representación XML de la Figura 10.1 para calcular la reunión externa por la izquierda de los elementos impositor con los elementos cuenta. (Consejo: se puede usar la cuantificación universal.)

10.8 Dese una consulta en XQuery para invertir el anidamiento de los datos del Ejercicio 10.2. Esto es, el nivel más externo del anidamiento de la salida debe tener los elementos correspondientes a los autores, y cada uno de estos elementos debe tener anidados los elementos correspondientes a todos los libros escritos por el autor.

10.9 Dese la DTD para una representación XML de la información de la Figura 2.29. Créese un tipo de elemento separado para representar cada relación, pero úsense ID e IDREF para implementar las claves primarias y externas.

10.10 Escríbanse consultas en XSLT y XQuery que devuelvan los elementos cliente con los elementos cuenta asociados anidados en los elementos cliente, dada la representación de la información bancaria usando ID e IDREFS de la Figura 10.8.

10.11 Dese un esquema relacional para representar la información bibliográfica como se especifica en el fragmento DTD en la Figura 10.13. El esquema relacional debe registrar el orden de los elementos autor. Se puede asumir que sólo los libros y artículos aparecen como elementos de nivel superior en los documentos XML.

10.12 Considérese el Ejercicio 10.11 y supóngase que los autores también pueden aparecer como elementos de nivel superior ¿Qué cambio habría que realizar en el esquema relacional?

10.13 Escríbanse las consultas en XQuery del fragmento DTD de bibliografía de la Figura 10.13 para realizar lo siguiente:

a. Encontrar todos los autores que tienen un libro y un artículo en el mismo año.

b. Mostrar los libros y artículos ordenados por años.

c. Mostrar los libros con más de un autor.

10.14 Mostrar la representación en árbol de los datos XML de la Figura 10.1 y la representación del árbol usando relaciones *nodos* e *hijo* descritas en el Apartado 10.6.1.

10.15 Considérese la siguiente DTD recursiva

```
<!DOCTYPE producto [
    <!ELEMENT producto (nombre, infocomponen-
        te*)>
    <!ELEMENT infocomponente (producto, cantidad)>
    <!ELEMENT nombre ( #PCDATA )>
    <!ELEMENT cantidad ( #PCDATA )>
]>
```

a. Dese un pequeño ejemplo de datos correspondientes a la DTD de arriba.

b. Muéstrese cómo hacer corresponder este DTD con un esquema relacional. Se puede asumir que los nombres de producto son únicos, esto es, cada vez que aparezca un producto, su estructura de componente será la misma.

```
<!DOCTYPE bibliografía [
    <!ELEMENT libro (título, autor+, año, editor, lugar?)>
    <!ELEMENT artículo (título, autor+, revista, año, número,
        volumen, páginas?)>
    <!ELEMENT autor (apellidos, nombre) >
    <!ELEMENT título ( #PCDATA )>
    ··· declaraciones PCDATA similares para año, editor, lugar,
        revista, año, número, volumen, páginas, apellidos y
        nombre
]>
```

FIGURA 10.13. DTD para los datos bibliográficos.

NOTAS BIBLIOGRÁFICAS

El sitio XML Cover Pages (www.oasis-open.org/cover/) contiene una serie de información XML, incluyendo introducciones a XML, estándares, publicaciones y software. El consorcio W3C (World Wide Web Consortium) actúa como el cuerpo de normas para normas relacionadas con la Web, incluyendo XML básico y todos los lenguajes relacionados con XML tales como XPath, XSLT y XQuery. Hay disponibles en www.w3c.org un gran número de informes técnicos que definen los estándares relacionados con XML.

Fernández et al. [2000] proporcionan un álgebra para XML. Quilt se describe en Chamberlin et al. [2000]. Sahuguet [2001] describe un sistema basado en el lenguaje Quilt para consultas XML. Deutsch et al. [1999b] describen el lenguaje XML-QL. La integración de consultas de palabras clave en XML se describe en Florescu et al. [2000]. La optimización de consultas para XML se describe en McHugh y Widom [1999]. Fernández y Morishima [2001] describen una evaluación eficiente de consultas XML en sistemas *middleware*. Otro trabajo sobre la consulta y manipulación de datos XML está en Chawathe [1999], Deutsch et al. [1999a] y Shanmugasundaram et al. [2000].

Florescu y Kossmann [1999], Kanne y Moerkotte [2000], y Shanmugasun-daram et al. [1999] describen el almacenamiento de datos XML. Schning [2001] describe una base de datos diseñada para XML. El soporte de XML para bases de datos comerciales está descrito en Banerjee et al. [2000], Cheng y Xu [2000], y Rys [2001]. Véanse los Capítulos 25 a 27 para más información sobre el soporte XML en bases de datos comerciales. El uso de XML para la integración de datos se describe en Liu et al. [2000], Draper et al. [2001], Baru et al. [1999], y Carey et al. [2000].

HERRAMIENTAS

Hay disponibles una serie de herramientas de uso público para tratar con XML. El sitio www.oasis-open.org/cover/ contiene enlaces a una serie de herramientas software para XML y XSL (incluyendo XSLT). Kweelt (disponible en http://db.cis.upenn.edu/Kweelt/) es un sistema de consulta XML disponible públicamente basado en el lenguaje Quilt.

PARTE

IV

ALMACENAMIENTO DE DATOS Y CONSULTAS

Aunque los sistemas de bases de datos proporcionan una visión de alto nivel de los datos, al final los datos se tienen que almacenar como bits en uno o varios dispositivos de almacenamiento. Una amplia mayoría de las bases de datos de hoy en día almacenan los datos en discos magnéticos y los extraen a la memoria del espacio principal para su procesamiento, o copian los datos en cintas y otros dispositivos de copia de seguridad para su almacenamiento en archivos. Las características físicas de los dispositivos de almacenamiento desempeñan un papel importante en el modo en que se almacenan los datos, en especial porque el acceso a un fragmento aleatorio de los datos en el disco resulta mucho más lento que el acceso a la memoria: los accesos al disco tardan decenas de milisegundos, mientras que el acceso a la memoria tarda una décima de microsegundo.

El Capítulo 11 comienza con una introducción a los medios físicos de almacenamiento, incluidos los mecanismos para minimizar la posibilidad de pérdidas de datos debidas a fallos. El capítulo describe a continuación el modo en que se asignan los registros a los archivos que, a su vez, se asignan a bits del disco. El almacenamiento y la recuperación de los objetos se trata también en el Capítulo 11.

Muchas consultas sólo hacen referencia a una pequeña parte de los registros de un archivo. Los índices son estructuras que ayudan a localizar rápidamente los registros deseados de una relación, sin examinar todos los registros. El índice de este libro de texto es un ejemplo, aunque, a diferencia de los índices de las bases de datos, está pensado para su empleo por personas. El Capítulo 12 describe varios tipos de índices utilizados en los sistemas de bases de datos.

Las consultas de los usuarios tienen que ejecutarse sobre el contenido de la base de datos, que reside en los dispositivos de almacenamiento. Suele resultar conveniente fraccionar las consultas en operaciones más pequeñas, que se corresponden aproximadamente con las operaciones del álgebra relacional. El Capítulo 13 describe el modo en que se procesan las consultas, presenta los algoritmos para la implementación de las operaciones individuales y esboza el modo en que las operaciones se ejecutan en sincronía para procesar una consulta.

Hay muchas maneras alternativas de procesar cada consulta, que pueden tener costes muy variables. La optimización de consultas hace referencia al proceso de hallar el método de coste mínimo para evaluar una consulta dada. El Capítulo 14 describe el proceso de optimización de consultas.

CAPÍTULO

11 ALMACENAMIENTO Y ESTRUCTURA DE ARCHIVOS

En los capítulos anteriores se han destacado los modelos de bases de datos de alto nivel. En el nivel *conceptual* o *lógico* se vieron las bases de datos, en el modelo relacional, como conjuntos de tablas. En realidad, el modelo lógico de las bases de datos es el nivel adecuado en el que se deben centrar los *usuarios*. Esto es por lo que el objetivo de un sistema de bases de datos es simplificar y facilitar el acceso a los datos; los usuarios del sistema no deben someterse sin necesidad alguna a la carga de los detalles físicos del desarrollo del sistema.

En este capítulo, así como en los Capítulos 12, 13 y 14, se desciende desde los niveles superiores y se describen varios métodos para implementar los modelos de datos y los lenguajes presentados en los capítulos anteriores. Se comienza con las características de los medios de almacenamiento subyacentes, como los sistemas de discos y de cintas. Posteriormente se definen varias estructuras de datos que permitirán un acceso rápido a los datos. Se estudian varias arquitecturas alternativas, cada una de ellas más adecuada a diferentes formas de acceder a los datos. La elección final de la estructura de datos hay que hacerla en función del uso que se espera dar al sistema y de las características de cada máquina concreta.

11.1. VISIÓN GENERAL DE LOS MEDIOS FÍSICOS DE ALMACENAMIENTO

En la mayor parte de los sistemas informáticos hay varios tipos de almacenamientos de datos. Estos medios de almacenamiento se clasifican según la velocidad con la que se puede acceder a los datos, por el coste de adquisición del medio por unidad de datos y por la fiabilidad del medio. Entre los medios disponibles habitualmente figuran:

- **Caché.** Caché es la forma de almacenamiento más rápida y costosa. La memoria caché es pequeña; su uso lo gestiona el hardware del sistema informático. No hay que preocuparse sobre la gestión del almacenamiento caché del sistema de bases de datos.
- **Memoria principal.** El medio de almacenamiento utilizado para operar con los datos disponibles es la memoria principal. Las instrucciones de la máquina de propósito general operan en la memoria principal. Aunque la memoria principal puede contener muchos megabites de datos, suele ser demasiado pequeña (o demasiado cara) para guardar toda la base de datos. El contenido de la memoria principal suele perderse si se produce un fallo del suministro eléctrico o una caída del sistema.
- **Memoria *flash*.** También conocida como *memoria sólo de lectura programable y borrable eléctricamente* (*Electrically Erasable Programmable Read-Only Memory*, EEPROM), la memoria *flash* se diferencia de la memoria principal en que los datos pueden sobrevivir a los fallos del suministro eléctrico. La lectura de los datos de la memoria *flash* tarda menos de cien nanosegundos (un nanosegundo es 0,001 milisegundo), lo que resulta aproximadamente igual de rápido que la lectura de los datos de la memoria principal. Sin embargo, la escritura de los datos en la memoria *flash* resulta más complicada (los datos pueden escribirse una sola vez, lo que tarda de cuatro a diez microsegundos, pero no se pueden sobrescribir de manera directa). Para sobreescribir la memoria que se ha escrito previamente hay que borrar simultáneamente todo un banco de memoria; entonces queda preparado para volver a escribir en él. Un inconveniente de la memoria *flash* es que sólo permite un número limitado de ciclos de borrado, que varía entre diez mil y un millón. La memoria *flash* se ha hecho popular como sustituta de los discos magnéticos para guardar pequeños volúmenes de datos (de cinco a diez megabytes) en los sistemas informáticos de coste reducido que se incluyen en otros dispositivos, como computadoras de bolsillo y en otros dispositivos electrónicos como cámaras digitales.
- **Almacenamiento en discos magnéticos.** El principal medio de almacenamiento a largo plazo de datos en conexión es el disco magnético. Generalmente se guarda en este tipo de discos toda la base de datos. Para tener acceso a los datos hay que tras-

ladarlos desde el disco a la memoria principal. Después de realizar la operación hay que escribir en el disco los datos que se han modificado.

El tamaño de los discos magnéticos actuales oscila entre unos pocos gigabytes y 80 gigabytes. Tanto el extremo superior como el inferior de este rango han ido creciendo a un ritmo del 50 por ciento anual, y se pueden esperar discos de mucha mayor capacidad cada año. El almacenamiento en disco puede aguantar los fallos del suministro eléctrico y caídas del sistema. Los dispositivos de almacenamiento en disco pueden fallar a veces y, en consecuencia, destruir los datos, pero tales fallos suelen producirse con mucha menos frecuencia que una caída del sistema.

- **Almacenamiento óptico.** La forma más popular de almacenamiento óptico es el disco compacto (*Compact Disk, CD*), que puede almacenar alrededor de 640 megabytes de datos, y el *disco de video digital* (*Digital Video Disk, DVD*), que puede almacenar 4,7 u 8,5 gigabytes de datos por cada cara del disco (o hasta 17 gigabytes en un disco de doble cara). Los datos se almacenan en un disco por medios ópticos y se leen mediante un láser. Los discos ópticos usados en los discos compactos de sólo lectura (CD-ROM) o en los discos de vídeo digital de sólo lectura (DVD-ROM) no se pueden escribir, pero se suministran con datos pregrabados.

  Hay versiones «grabar una vez» de los discos compactos (llamada CD-R) y de los discos de vídeo digital (llamada DVD-R), que se pueden escribir sólo una vez; estos disco también se denominan **de escritura única y lectura múltiple** (*Write-Once, Read-Only Memory*, WORM). También hay versiones «escribir varias veces» de los discos compactos (llamada CD-RW) y de los discos de vídeo digital (DVD-RW y DVD-RAM), que permiten escribir varias veces. Los discos compactos grabables son dispositivos de almacenamiento magnetoópticos que usan medios ópticos para leer datos codificados magnéticamente. Estos discos son útiles para el almacenamiento de archivos así como para la distribución de datos.

  Los *cambiadores automáticos* (*jukebox*) contienen unas cuantas unidades y numerosos discos que pueden cargarse de manera automática en una de las unidades (mediante un brazo robotizado) a petición del usuario.

- **Almacenamiento en cinta.** El almacenamiento en cinta se utiliza principalmente para copias de seguridad y datos de archivo. Aunque la cinta magnética es mucho más barata que los discos, el acceso a los datos resulta mucho más lento, ya que el acceso a la cinta debe ser secuencial desde su comienzo. Por este motivo, el almacenamiento se denomina almacenamiento de **acceso secuencial**. En cambio, el almacenamiento en disco se denomina almacenamiento de acceso directo porque es posible leer datos desde cualquier ubicación del disco.

  Las cintas tienen una capacidad elevada (actualmente hay disponibles cintas de 40 a 300 gigabytes) y pueden retirarse de la unidad de lectura, lo que facilita un almacenamiento de coste reducido para archivos. Los cambiadores automáticos de cinta se utilizan para guardar conjuntos de datos excepcionalmente grandes, como los datos de sensores remotos de los satélites, que pueden alcanzar cientos de terabytes (10^{12} bytes) o incluso petabytes (10^{15} bytes) de datos.

Los diferentes medios de almacenamiento pueden organizarse en una jerarquía (Figura 11.1) de acuerdo con su velocidad y su coste. Los niveles superiores son de coste elevado, pero rápidos. A medida que se desciende por la jerarquía el coste por bit disminuye, mientras que el tiempo de acceso aumenta. Este compromiso es razonable; si un sistema de almacenamiento dado fuera a la vez más rápido y menos costoso que otro (siendo iguales las demás propiedades) no habría ninguna razón para utilizar la memoria más lenta y más costosa. De hecho, muchos dispositivos de almacenamiento primitivos, incluyendo la cinta de papel y las memorias de núcleos de ferrita, se hallan relegados a los museos ahora que la cinta magnética y la memoria de semiconductores se han vuelto más rápidas y económicas. Las propias cintas magnéticas se utilizaron para guardar los datos activos cuando los discos resultaban costosos y tenían una capacidad de almacenamiento reducida. Hoy en día casi todos los datos activos se almacenan en discos, excepto en los raros casos en que se guardan en cambiadores automáticos de cinta u ópticos.

Los medios de almacenamiento más rápidos (por ejemplo, caché y memoria principal) se denominan **almacenamiento primario**. Los medios del siguiente

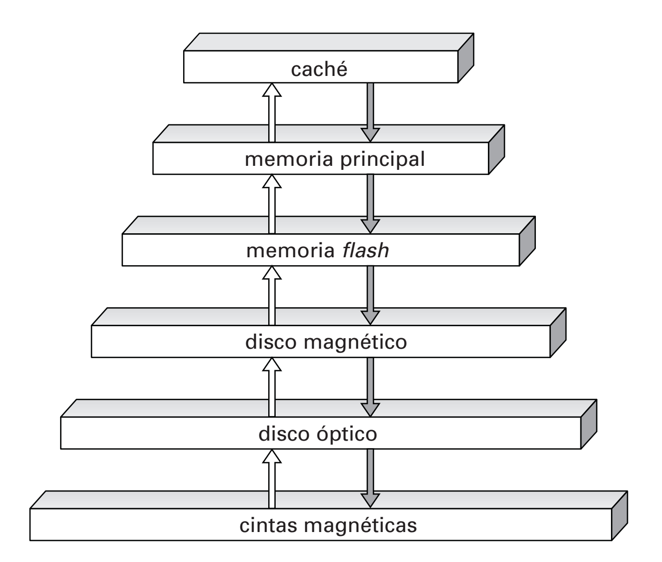


FIGURA 11.1. Jerarquía de dispositivos de almacenamiento.

nivel de la jerarquía (por ejemplo, los discos magnéticos) se conocen como **almacenamiento secundario** o **almacenamiento en conexión**. Los medios del nivel inferior de la jerarquía —por ejemplo, cinta magnética y los cambiadores automáticos de discos ópticos— se denominan **almacenamiento terciario** o **almacenamiento sin conexión**.

Además de la velocidad y del coste de los diferentes sistemas de almacenamiento está también el asunto de la volatilidad del almacenamiento. El **almacenamiento volátil** pierde su contenido cuando se suprime el suministro eléctrico del dispositivo. En la jerarquía mostrada en la Figura 11.1 los sistemas de almacenamiento desde la memoria principal hacia arriba son volátiles, mientras que los sistemas de almacenamiento por debajo de la memoria principal son no volátiles. En ausencia de los costosos sistemas de baterías y de sistemas de generadores de reserva, los datos deben escribirse en **almacenamiento no volátil** por razones de seguridad. Se volverá a este asunto en el Capítulo 17.

11.2. DISCOS MAGNÉTICOS

Los discos magnéticos constituyen el principal medio de almacenamiento secundario en los sistemas informáticos modernos. Las capacidades de los discos han estado creciendo alrededor del 50 por ciento anual, pero los requisitos de las grandes aplicaciones también han estado creciendo muy rápido, en algunos casos más que la tasa de crecimiento de las capacidades de los discos. Una base de datos comercial grande típica puede necesitar centenares de discos.

11.2.1. Características físicas de los discos

Físicamente, los discos son relativamente sencillos (Figura 11.2). Cada **plato** del disco tiene una forma circular plana. Sus dos superficies están cubiertas por un material magnético y la información se graba en ellas. Los platos están hechos de metal rígido o de vidrio y están cubiertos (generalmente por los dos lados) con material magnético para grabaciones. Estos discos magnéticos se denominan **discos rígidos** para distinguirlos de los **disquetes**, que están hechos de material flexible.

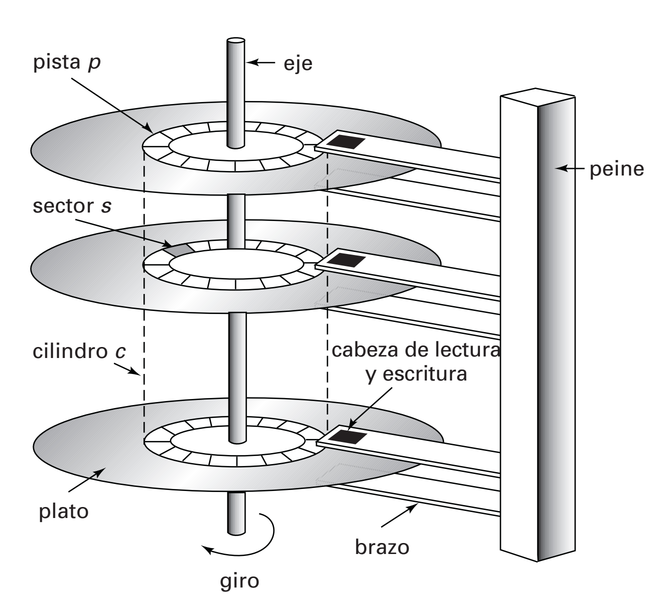


FIGURA 11.2. Mecanismo de disco de cabezas móviles.

Mientras se está utilizando el disco, un motor lo hace girar a una velocidad constante elevada (generalmente 60, 90 o 120 revoluciones por segundo, pero también hay disponibles discos girando a 250 revoluciones por segundo). Hay una cabeza de lectura y escritura ubicada justo encima de la superficie del plato. La superficie del disco se divide a efectos lógicos en **pistas**, que se subdividen en **sectores**. Un **sector** es la unidad mínima de información que puede leerse o escribirse en el disco. En los discos disponibles actualmente, los tamaños de sector son normalmente de 512 bytes; hay alrededor de 16.000 pistas en cada disco y de dos a cuatro platos por disco. Las pistas internas (más cercanas al eje) son de menor longitud, y en los disco de la generación actual, las pistas exteriores contienen más sectores que las pistas internas; normalmente hay alrededor de 200 sectores por pista en las pistas internas y alrededor de 400 sectores por pistas en las pistas externas. Estos números pueden variar entre diferentes modelos; los modelos de mayor capacidad tienen más sectores por pista y más pistas en cada plato.

La **cabeza de lectura y escritura** guarda magnéticamente la información en los sectores en forma de inversiones de la dirección de magnetización del material magnético.

Cada cara de un plato del disco tiene una cabeza de lectura y escritura que se desplaza por el plato para tener acceso a las diferentes pistas. Un disco suele contener muchos platos y las cabezas de lectura y escritura de todas las pistas están montadas en un solo dispositivo denominado **brazo del disco** y se desplazan conjuntamente. El conjunto de los platos del disco montados sobre un eje y las cabezas montadas en el brazo del disco se denomina **dispositivo cabeza-disco**. Dado que las cabezas de todos las platos se desplazan conjuntamente, cuando la cabeza de un plato se halle en la pista *i*-ésima, las cabezas de todos las demás platos también se encontrarán en la pista *i*-ésima de sus platos respectivos. Por consiguiente, las pistas *i*-ésimas de todos los platos se denominan conjuntamente **cilindro** *i*-ésimo.

Actualmente, los discos con un diámetro de plato de 3 $^{1}/_{2}$ pulgadas dominan el mercado. Tienen un menor coste y tiempos de búsqueda más cortos (debido a las menores distancias de búsqueda) que los discos de tamaño mayor (de hasta 14 pulgadas) que fueron habituales en el pasado, y proporcionan una alta capacidad de almacenamiento.

Las cabezas de lectura y escritura se mantienen tan próximas como sea posible a la superficie de los discos para aumentar la densidad de grabación. A menudo la cabeza flota o vuela a sólo micras de la superficie del disco; las revoluciones del disco crean una pequeña brisa y el dispositivo de cabezas se manufactura de forma que la brisa mantenga la cabeza flotando sobre la superficie del disco. Como la cabeza flota tan cercana a la superficie, los platos deben estar elaborados esmeradamente para que sean lisos. Los choques de las cabezas pueden suponer un problema. Si la cabeza contacta con la superficie del disco, arrancará del disco el medio de grabación, destruyendo los datos que hubiera allí. Generalmente, la cabeza que toca la superficie hace que el material arrancado flote en el aire y se coloque entre las cabezas y sus platos, lo que causa más choques. En circunstancias normales un choque de las cabezas da lugar a que falle todo el disco y haya que sustituirlo. Los dispositivos de disco de la generación actual usan una fina capa de metal magnético como medio de grabación. Son menos susceptibles de fallar a causa de choques de las cabezas que los antiguos discos cubiertos de óxido.

Un *disco de cabezas fijas* tiene una cabeza diferente para cada pista. Esta disposición permite a la computadora cambiar rápidamente de pista a pista sin tener que desplazar el dispositivo de las cabezas, pero necesita gran número de cabezas, lo que hace al dispositivo excesivamente costoso. Algunos sistemas de discos tienen varios brazos de disco, lo que permite que se tenga acceso simultáneamente a más de una pista del mismo plato. Los discos de cabezas fijas y los discos con varios brazos se utilizaban en grandes sistemas de alto rendimiento, pero ya no se fabrican.

Un **controlador de disco** actúa como interfaz entre el sistema informático y el hardware concreto de la unidad de disco. Acepta las órdenes de alto nivel para leer o escribir en un sector e inicia las acciones, como desplazar el brazo del disco a la pista adecuada y leer o escribir realmente los datos. Los controladores de disco también añaden **comprobación de suma** a cada sector en el que se escribe; la comprobación de suma se calcula a partir de los datos escritos en el sector. Cuando se vuelve a leer el sector, se vuelve a calcular la suma a partir de los datos recuperados y se compara con la suma de comprobación guardada; si los datos se han deteriorado resulta muy probable que la suma de comprobación recién calculada no coincida con la guardada. Si se produce un error de este tipo, el controlador volverá a intentar varias veces la lectura; si el error sigue produciéndose, el controlador indicará un fallo de lectura.

Otra labor interesante llevada a cabo por los controladores de disco es la **reasignación de los sectores dañados**. Si el controlador detecta que un sector está dañado cuando se da formato al disco por primera vez, o cuando se realiza un intento de escribir en el sector, puede reasignar lógicamente el sector a una ubicación física diferente (escogida de entre un grupo de sectores extra preparado con esta finalidad). La reasignación se anota en disco o en memoria no volátil y la escritura se realiza en la nueva ubicación.

En la Figura 11.3 se muestra la manera en que los discos se conectan a un sistema informático. Al igual que otras unidades de almacenamiento, los discos se conectan a un sistema informático o a un controlador mediante una conexión de alta velocidad. En los sistemas de disco modernos, las funciones de menor nivel del controlador de disco, como el control del brazo, el cálculo y verificación de la comprobación de suma y la reasignación de los sectores dañados se implementan en la unidad de disco.

La interfaz **ATA (AT Attachment)** (que es una versión más rápida que la **interfaz electrónica de dispositivos integrados** [IDE, Integrated Drive Electronics] usada antiguamente en los PC de IBM) y la **interfaz de conexión para sistemas informáticos pequeños** (*SCSI, Small Computer-System Interconnect Interface*, pronunciado «escasi») se usan habitualmente para conectar los discos con las computadoras personales y las estaciones de trabajo. Los grandes sistemas y los siste-

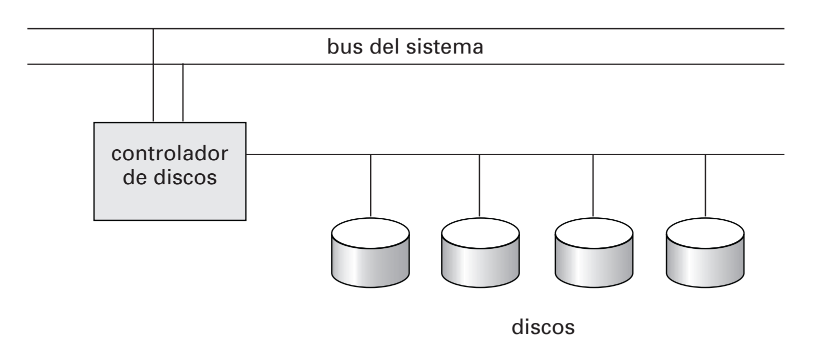


FIGURA 11.3. Subsistema de disco.

mas servidores suelen disponer de una interfaz más rápida y cara, como las versiones de alta capacidad de la interfaz SCSI, y la interfaz Fibre Channel.

Los discos se conectan por lo general directamente al controlador de disco mediante cables, pero también pueden estar situados en una ubicación remota y estar conectados mediante una red de alta velocidad al controlador de disco. En una arquitectura de **red de área de almacenamiento** (Storage-Area Network, SAN), se conecta un gran número de discos mediante una red de alta velocidad a varias computadoras servidoras. Los discos generalmente se organizan localmente usando una técnica de organización del almacenamiento denominada «disposición redundante de discos independientes (Redundant Array of Independent Disks, RAID)» (descritos posteriormente en el Apartado 11.3) para dar a los servidores una vista lógica de un disco de gran tamaño y muy fiable. El controlador y el disco usan las interfaces SCSI y Fibre Channel para comunicarse entre sí, aunque puedan estar separados por una red. El acceso remoto a los discos también significa que los discos que contienen datos importantes se pueden guardar en una habitación del servidor central donde se pueden supervisar y mantener por los administradores del sistema, en lugar de estar esparcidos en diferentes lugares de la organización.

11.2.2. Medidas del rendimiento de los discos

Las principales medidas de la calidad de un disco son la capacidad, el tiempo de acceso, la velocidad de transferencia de datos y la fiabilidad.

El **tiempo de acceso** es el tiempo transcurrido desde que se formula una solicitud de lectura o de escritura hasta que comienza la transferencia de datos. Para tener acceso (es decir, para leer o escribir) a los datos en un sector dado del disco, primero se debe desplazar el brazo para que se ubique sobre la pista correcta y luego hay que esperar a que el sector aparezca bajo el brazo por acción de la rotación del disco. El tiempo para volver a ubicar el brazo se denomina **tiempo de búsqueda** y aumenta con la distancia que deba recorrer el brazo. Los tiempos de búsqueda típicos varían de dos a treinta milisegundos, en función de la distancia de la pista a la posición inicial del brazo. Los discos de menor tamaño tienden a tener tiempos de búsqueda menores, dado que la cabeza tiene que recorrer una distancia menor.

El **tiempo medio de búsqueda** es la media de los tiempos de búsqueda medido en una sucesión de solicitudes aleatorias (uniformemente distribuidas). Si todas las pistas tienen el mismo número de sectores y despreciando el tiempo requerido para que la cabeza inicie su movimiento y lo detenga, se puede demostrar que el tiempo medio de búsqueda es un tercio del peor de los tiempos de búsqueda posibles. Teniendo en cuenta estos factores, el tiempo medio de búsqueda es alrededor de la mitad del tiempo máximo de búsqueda. Los tiempos medios de búsqueda varían actualmente entre cuatro y diez milisegundos, dependiendo del modelo de disco.

Una vez ha tenido lugar la búsqueda, el tiempo que se pasa esperando a que el sector al que hay que tener acceso aparezca bajo la cabeza se denomina **tiempo de latencia rotacional**. Las velocidades rotacionales típicas de los discos actuales varían de 5.400 rotaciones por minuto (90 rotaciones por segundo) hasta 15.000 rotaciones por minuto (250 rotaciones por segundo) o, lo que es lo mismo, de 4 a 11,1 milisegundos por rotación. En media hace falta la mitad de una rotación del disco para que aparezca bajo la cabeza el comienzo del sector deseado. Por tanto, el **tiempo de latencia medio** del disco es la mitad del tiempo empleado en una rotación completa del disco.

El tiempo de acceso es la suma del tiempo de búsqueda y del tiempo de latencia y varía de 8 a 20 milisegundos. Una vez que se ha ubicado bajo la cabeza el primer sector de datos, comienza su transferencia. La **velocidad de transferencia de datos** es la velocidad a la que se pueden recuperar o guardar datos en el disco. Los sistemas de disco actuales anuncian que permiten velocidades máximas de transferencia de 25 a 40 megabytes por segundo, aunque las velocidades de transferencia reales pueden ser significativamente menores, alrededor de 4 a 8 megabytes por segundo.

La última de las medidas de los discos utilizadas con frecuencia es el **tiempo medio entre fallos**, que es una medida de la fiabilidad del disco. El tiempo medio entre fallos de un disco (o de cualquier otro sistema) es la cantidad de tiempo que, en media, se puede esperar que el sistema funcione de manera continua sin tener ningún fallo. De acuerdo con los anuncios de los fabricantes, el tiempo medio entre fallos de los discos actuales varía de 30.000 a 1.200.000 horas (de 3,4 a 136 años). En la práctica, el tiempo medio entre fallos anunciado se calcula en términos de la probabilidad de fallo cuando el disco es nuevo (este escenario significa que dados 1.000 discos relativamente nuevos, si el tiempo medio entre fallos es 1.200.000 horas, uno de ellos fallará en media en 1.200 horas). Un tiempo medio entre fallos de 1.200.000 horas no implica que se espere que el disco vaya a funcionar 136 años. La mayoría de los discos tienen un periodo de vida esperado de cinco años, y tienen significativamente más fallos cuando son algunos años más viejos.

Puede haber varios discos compartiendo una interfaz de discos. La norma de la interfaz ATA-4 ampliamente usada (también conocida como Ultra-DMA) soporta velocidades de transferencia de 33 megabytes por segundo, mientras que ATA-5 soporta 66 megabytes por segundo. SCSI-3 (Ultra 2 wide SCSI) soporta 40 megabytes por segundo, mientras que la interfaz más cara Fibre Channel soporta hasta 256 megabytes por segundo. La velocidad de transferencia de la interfaz se comparte entre todos los discos conectados a la interfaz.

11.2.3. Optimización del acceso a los bloques del disco

Las solicitudes de E/S al disco las generan tanto el sistema de archivos como el gestor de la memoria virtual que se halla en la mayor parte de los sistemas operativos. Cada solicitud especifica la dirección del disco a la que hay que hacer referencia; esa dirección está en la forma de un *número de bloque*. Un **bloque** es una secuencia continua de sectores de una sola pista de un plato. Los tamaños de los bloques varían de 512 bytes a varios kilobytes. Los datos se transfieren entre el disco y la memoria principal en unidades de bloques. Los niveles inferiores del gestor del sistema de archivos transforman las direcciones de los bloques en cilindro, superficie y número de sector del nivel de hardware.

Dado que el acceso a los datos del disco es varios órdenes de magnitud más lento que el acceso a la memoria principal, se ha prestado mucha atención a la mejora de la velocidad de acceso a los bloques del disco. La utilización de memoria intermedia para bloques en la memoria para satisfacer las solicitudes futuras se discute en el Apartado 11.5. Aquí se discuten otras técnicas.

- **Planificación.** Si hay que transferir varios bloques de un cilindro desde el disco a la memoria principal puede que se logre disminuir el tiempo de acceso solicitando los bloques en el orden en el que pasarán por debajo de las cabezas. Si los bloques deseados se hallan en cilindros diferentes resulta ventajoso solicitar los bloques en un orden que minimice el movimiento del brazo del disco. Los algoritmos de **planificación del brazo del disco** intentan ordenar el acceso a las pistas de manera que se aumente el número de accesos que puede procesarse. Un algoritmo utilizado con frecuencia es el **algoritmo del ascensor**, que funciona de manera parecida a muchos ascensores. Supóngase que, inicialmente, el brazo se desplaza desde la pista más interna hacia el exterior del disco. Bajo el control del algoritmo del ascensor, en cada pista para la que hay una solicitud de acceso el brazo se detiene, atiende las solicitudes para la pista y continúa desplazándose hacia el exterior hasta que no queden solicitudes pendientes para las pistas más externas. En este punto el brazo cambia de dirección, se desplaza hacia el interior y vuelve a detenerse en cada pista para las que haya solicitudes hasta alcanzar una pista en la que no haya solicitudes para pistas más cercanas al centro del disco. En ese momento cambia de dirección e inicia un nuevo ciclo. Los controladores de disco suelen realizar la labor de reordenar las solicitudes de lectura para mejorar el rendimiento, dado que conocen perfectamente la organización de los bloques del disco, la posición rotacional de los platos y la posición del brazo.
- **Organización de archivos.** Para reducir el tiempo de acceso a los bloques se pueden organizar los bloques del disco de una manera que se corresponda fielmente con la forma en que se espera tener acceso a los datos. Por ejemplo, si se espera tener acceso secuencial a un archivo, en teoría se deberían guardar secuencialmente en cilindros adyacentes todos los bloques del archivo. Los sistemas operativos más antiguos, como los sistemas operativos de IBM para grandes sistemas proporcionaban a los programadores un control detallado de la ubicación de los archivos, lo que permitía a los programadores reservar un conjunto de cilindros para guardar un archivo. Sin embargo, este control supone una carga para el programador o el administrador del sistema que debe decidir, por ejemplo, los cilindros que ha de asignar a un archivo y puede exigir una reorganización costosa si se insertan o borran datos del archivo.

  Los sistemas operativos posteriores, como Unix y los sistemas operativos para computadoras personales ocultan a los usuarios la organización del disco y gestionan la asignación de manera interna. Sin embargo, en el transcurso del tiempo, un archivo secuencial puede quedar **fragmentado**; es decir, sus bloques pueden quedar dispersos por el disco. Para reducir la fragmentación, el sistema puede hacer una copia de seguridad de los datos del disco y restaurar todo el disco. La operación de restauración vuelve a escribir de manera contigua (o casi) los bloques de cada archivo. Algunos sistemas (como MS-DOS) disponen de utilidades que examinan el disco y luego desplazan los bloques para reducir la fragmentación. Los aumentos de rendimiento obtenidos con estas técnicas pueden ser elevados, pero no se suele poder utilizar el sistema mientras se están ejecutando estas utilidades.

- **Memoria intermedia de escritura no volátil.** Dado que el contenido de la memoria se pierde durante los fallos de suministro eléctrico, hay que guardar en disco la información sobre las actualizaciones de las bases de datos para que superen las posibles caídas del sistema. Por tanto, el rendimiento de las aplicaciones de bases de datos sensibles a las actualizaciones, como los sistemas de procesamiento de transacciones, dependen mucho de la velocidad de escritura en el disco.

  Se puede utilizar **memoria no volátil de acceso aleatorio** (RAM no volátil) para acelerar la escritura en el disco de manera drástica. El contenido de la RAM no volátil no se pierde durante un fallo del suministro eléctrico. Una manera habitual de implementar la RAM no volátil es utilizar RAM alimentada por baterías. La idea es que, cuando el sistema de bases de datos (o el sistema operativo) solicita que se escriba un bloque en el disco, el controlador del disco escriba el bloque en una memoria intermedia de RAM no volátil y comunique de manera inmediata al sistema ope-

rativo que la escritura se completó con éxito. El controlador escribe los datos en su destino en el disco en cualquier momento en que el disco no tenga otras solicitudes o cuando la memoria intermedia de RAM no volátil se llene. Cuando el sistema de bases de datos solicita la escritura de un bloque sólo percibe un retraso si la memoria intermedia de RAM no volátil se encuentra llena. Durante la recuperación de una caída del sistema se vuelven a escribir en el disco todas las escrituras que se hallan pendientes en la memoria intermedia de RAM no volátil.

El grado en el que la RAM no volátil aumenta el rendimiento se ilustra en el siguiente ejemplo. Supóngase que las solicitudes de escritura se reciben de manera aleatoria y que el disco se halla ocupado el 90 por ciento del tiempo como media[1]. Si se dispone de una memoria intermedia de RAM no volátil de cincuenta bloques, las solicitudes de escritura sólo encontrarán la memoria intermedia llena, en media, una vez por minuto (y tendrán, por tanto, que esperar a que concluya un proceso de escritura en el disco). Duplicar la memoria intermedia a cien bloques resulta en encontrar la memoria intermedia llena sólo una vez por hora. Por tanto, en la mayoría de los casos, las escrituras a disco se pueden ejecutar sin que el sistema de bases de datos espere debido a una búsqueda o a la latencia rotacional.

- **Disco de registro histórico.** Otro enfoque para reducir las latencias de escritura es utilizar un *disco de registro histórico* (de manera muy parecida a la memoria intermedia RAM no volátil). Todos los accesos al disco de registro histórico son secuenciales, lo que elimina principalmente el tiempo de búsqueda y, así, pueden escribirse simultáneamente varios bloques consecutivos, lo que hace que los procesos de escritura en el disco sean varias veces más rápidos que los procesos de escritura aleatorios. Igual que ocurría anteriormente también hay que escribir los datos en su ubicación verdadera en el disco, pero este proceso de escritura puede llevarse a cabo sin que el sistema de bases de datos tenga que esperar a que se complete. Más aún, el disco de registro histórico puede reordenar las escrituras para reducir el movimiento del brazo. Si el sistema cae antes de que se hayan realizado algunas escrituras en la ubicación real del disco, cuando el sistema se recupere lee el disco de registro histórico para encontrar las escrituras que no se han realizado y entonces las completa.

  Los sistemas de archivo que soportan los discos de registro histórico mencionados se denominan **sistemas de archivos de diario**. Los sistemas de archivos de diario se pueden implementar incluso sin un disco de registro histórico aislado, guardando los datos y el registro histórico en el mismo disco. Al hacerlo así se reduce el coste económico a expensas de un menor rendimiento.

  El **sistema de archivos basado en registro histórico** es una versión extrema del enfoque del disco de registro histórico. Los datos no se vuelven a escribir en su ubicación original en el disco; en su lugar, el sistema de archivos hace un seguimiento del lugar del disco de registro histórico en que se escribieron los bloques más recientemente y los recupera desde esa ubicación. El propio disco de registro histórico se compacta de manera periódica, por lo que se pueden eliminar los procesos de escritura antiguos que se han sobrescrito posteriormente. Este enfoque mejora el rendimiento de los procesos de escritura pero genera un elevado grado de fragmentación de los archivos que se actualizan con frecuencia. Como se señaló anteriormente, esa fragmentación aumenta el tiempo de búsqueda en la lectura secuencial de los archivos.

11.3. RAID

Los requisitos de almacenamiento de datos de algunas aplicaciones (en particular las aplicaciones Web, de bases de datos y multimedia) han estado creciendo tan rápidamente que se necesita un gran número de discos para almacenar sus datos, incluso aunque las capacidades de los discos hayan estado creciendo muy rápidamente.

Tener un gran número de discos en un sistema presenta oportunidades para mejorar la velocidad a la que se pueden leer o escribir los datos si los discos funcionan en paralelo. El paralelismo se puede usar para realizar varias lecturas o escrituras independientes simultáneamente. Además, esta configuración ofrece la posibilidad de mejorar la fiabilidad del almacenamiento de datos, ya que se puede guardar información repetida en varios discos. Por tanto, el fallo de un disco no provoca una pérdida de datos.

Para abordar los problemas de rendimiento y de fiabilidad se han propuesto varias técnicas de organización de discos, denominadas colectivamente **disposición redundante de discos independientes** (**RAIDs - Redundant Array of Independent Disks**), para conseguir un rendimiento y fiabilidad mejorados.

[1] Para el lector interesado en la estadística se asume una distribución de Poisson para las llegadas. Las tasas exactas de llegada y de servicio no se necesitan, dado que el uso del disco proporciona suficiente información para estos cálculos.

En el pasado, los diseñadores de sistemas vieron los sistemas de almacenamiento compuestos de discos pequeños y de bajo coste como una alternativa económicamente efectiva a los discos grandes y caros; el coste por megabyte de los discos más pequeños era menor que el de los discos mayores. De hecho, la I de RAID, que ahora representa *independientes*, originalmente representaba *económicos* (*inexpensive*). Hoy en día, sin embargo, todos los discos son pequeños físicamente, y los discos de gran capacidad tienen realmente un menor coste por megabyte. Los sistemas RAID se usan por su mayor fiabilidad y por su mayor velocidad de transferencia de datos, más que por motivos económicos.

11.3.1. Mejora de la fiabilidad mediante la redundancia

Considérese en primer lugar la fiabilidad. La posibilidad de que algún disco de una disposición de *N* discos falle es mucho más elevada que la posibilidad de que un único disco concreto falle. Supóngase que el tiempo medio entre fallos de un solo disco es de cien mil horas o, aproximadamente, once años. Por tanto, el tiempo medio entre fallos de un disco de una disposición de cien discos será de 100.000/100 = 1.000 horas, o 42 días, ¡lo que no es mucho! Si sólo se guarda una copia de los datos, cada fallo de un disco dará lugar a la pérdida de una cantidad de datos significativa (como se discutió anteriormente en el Apartado 11.2.1). Una tasa de pérdida de datos tan elevada resulta inaceptable.

La solución al problema de la fiabilidad es introducir la **redundancia**; es decir, se guarda información adicional que normalmente no se necesita pero que puede utilizarse en caso de fallo de un disco para reconstruir la información perdida. Por tanto, incluso si falla un disco, los datos no se pierden, por lo que el tiempo medio efectivo entre fallos aumenta, siempre que se cuenten sólo los fallos que den lugar a pérdida de datos o a su no disponibilidad.

El enfoque más sencillo (pero el más costoso) para la introducción de la redundancia es duplicar todos los discos. Esta técnica se denomina **creación de imágenes** (o, a veces, *creación de sombras*). Un disco lógico consiste, por tanto, en dos discos físicos y cada proceso de escritura se lleva a cabo en ambos discos. Si uno de los discos falla se pueden leer los datos del otro. Los datos sólo se perderán si falla el segundo disco antes de que se repare el primero que falló.

El tiempo medio entre fallos (donde fallo es la pérdida de datos) de un disco con imagen depende del tiempo medio entre fallos de cada disco y del **tiempo medio de reparación**, que es el tiempo que se tarda (en media) en sustituir un disco averiado y en restaurar sus datos. Supóngase que los fallos de los dos discos son *independientes*; es decir, no hay conexión entre el fallo de un disco y el del otro. Por tanto, si el tiempo medio entre fallos de un solo disco es de cien mil horas y el tiempo medio de reparación es de diez horas el **tiempo medio entre pérdidas de datos** de un sistema de discos con imagen es $100.000^2/(2 \times 10) = 500 \times 10^6$ horas, o ¡57.000 años! (aquí no se entra en detalle en los cálculos; se proporcionan en las referencias de las notas bibliográficas).

Hay que tener en cuenta que la suposición de la independencia de los fallos de los discos no resulta válida. Los fallos en el suministro eléctrico y los desastres naturales como los terremotos, los incendios y las inundaciones pueden dar lugar a daños simultáneos de ambos discos. A medida que los discos envejecen, la probabilidad de fallo aumenta, lo que aumenta la posibilidad de que falle el segundo disco mientras se repara el primero. A pesar de todas estas consideraciones, sin embargo, los sistemas de discos con imagen ofrecen una fiabilidad mucho más elevada que los sistemas de discos únicos. Hoy en día están disponibles sistemas de discos con imagen con un tiempo medio entre pérdidas de datos de entre quinientas mil y un millón de horas, o de cincuenta y cinco a ciento diez años.

Los fallos en el suministro eléctrico son una causa especial de preocupación, dado que tienen lugar mucho más frecuentemente que los desastres naturales. Los fallos en el suministro eléctrico no son un problema si no se está realizando ninguna transferencia de datos al disco cuando tienen lugar. Sin embargo, incluso con la creación de imágenes de los discos, si los procesos de escritura se hallan en curso en el mismo bloque en ambos discos y el suministro eléctrico falla antes de que ambos bloques estén escritos completamente, los dos bloques pueden quedar en un estado inconsistente. La solución a este problema es escribir una copia en primer lugar y luego la otra, de modo que siempre sea consistente una de las copias. Hacen falta algunas acciones adicionales cuando se vuelve a iniciar el sistema después de un fallo en el suministro eléctrico para recuperar los procesos de escritura incompletos. Este asunto se examina en el Ejercicio 11.4.

11.3.2. Mejora del rendimiento mediante el paralelismo

Considérense ahora las ventajas del acceso en paralelo a varios discos. Con la creación de imágenes de los discos la velocidad a la que las solicitudes de lectura pueden procesarse se duplica, dado que las solicitudes de lectura pueden enviarse a cualquiera de los discos (siempre y cuando los dos discos de la pareja estén operativos, como es el caso habitual). La velocidad de transferencia de cada proceso de lectura es la misma que en los sistemas de discos únicos, pero el número de procesos de lectura por unidad de tiempo se ha duplicado.

Con varios discos también se puede mejorar la velocidad de transferencia **distribuyendo los datos** entre varios discos. En su forma más sencilla la distribución de datos consiste en dividir los bits de cada byte entre varios discos; esta distribución se denomina **distribución en el nivel de bit**. Por ejemplo, si se dispone de una disposición de ocho discos se puede escribir el bit *i* de cada byte en el disco *i*. La disposición de ocho dis-

cos puede tratarse como un solo disco con sectores que tienen ocho veces el tamaño normal y, lo que es más importante, que tienen ocho veces la velocidad de acceso. En una organización así cada disco toma parte en todos los accesos (de lectura o de escritura) por lo que el número de accesos que pueden procesarse por segundo es aproximadamente el mismo que en un solo disco, pero cada acceso puede leer ocho veces tantos datos por unidad de tiempo como con un solo disco. La distribución en el nivel de bit puede generalizarse a cualquier número de discos que sea múltiplo o divisor de ocho. Por ejemplo, si se utiliza una disposición de cuatro discos, los bits i y $4 + i$ de cada byte irán al disco i.

La *distribución en el nivel de bloque* reparte los bloques entre varios discos. Trata la disposición de discos como un único y gran disco, y proporciona números lógicos a los bloques; se asume que los números de bloque comienzan en 0. Con una disposición de n discos, la distribución en el nivel de bloque asigna el bloque lógico i de la disposición de discos al disco $(i \bmod n) + 1$; usa el bloque físico $\lfloor i/n \rfloor$ – ésimo del disco para almacenar el bloque lógico i. Por ejemplo, con ocho discos, el bloque lógico 0 se almacena el bloque físico 0 del disco 1, mientras que el bloque lógico 11 se almacena en el bloque físico 1 del disco 4. Al leer un archivo grande, la distribución en el nivel de bloque busca n bloques en un instante en paralelo en los n discos, dando una gran velocidad de transferencia para grandes lecturas. Cuando se lee un único bloque, la velocidad de transferencia de datos es igual que en un disco, pero los restantes n - 1 discos están libres de realizar cualquier otra acción.

La distribución en el nivel de bloque es la forma de distribución de datos más usada. También son posibles otros niveles de distribución, como los bytes de cada sector o los sectores de cada bloque.

En resumen, hay dos objetivos principales para el paralelismo en un sistema de discos:

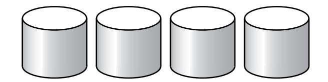
Equilibrar la carga de varios accesos de pequeño tamaño (accesos a bloque) de manera que la productividad de ese tipo de accesos aumente.

Convertir en paralelos los accesos de gran tamaño para que su tiempo de respuesta se reduzca.

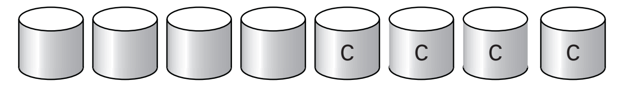
11.3.3. Niveles de RAID

La creación de imágenes proporciona gran fiabilidad pero resulta costosa. La distribución proporciona velocidades de transferencia de datos elevadas pero no mejora la fiabilidad. Se han propuesto esquemas alternativos para proporcionar redundancia a bajo costo utilizando la idea de la distribución de los discos combinada con los bits de «paridad» (que se describen a continuación). Estos esquemas tienen diferentes compromisos entre el coste y el rendimiento. Los esquemas se clasifican en niveles denominados **niveles de RAID**, como se muestra en la Figura 11.4 (en la figura, P indica los bits para la corrección de errores y C indica una segunda copia de los datos). En todos los casos descritos en la figura se guardan cuatro discos de datos y los discos adicionales para guardar la información redundante para la recuperación en caso de fallo.

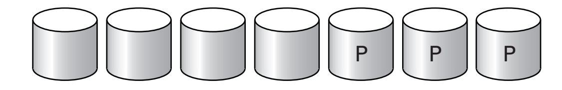
- **RAID de nivel 0** se refiere a disposiciones de discos con distribución en el nivel de bloque pero sin redundancia (como la creación de imágenes o los bits de paridad). En la Figura 11.4a se muestra una disposición de tamaño cuatro.
- **RAID de nivel 1** se refiere a la creación de imágenes del disco con distribución de bloques. En la Figura 11.4b se muestra una organización con imagen que tiene cuatro discos con datos.
- **RAID de nivel 2** también se conoce como organización de códigos de corrección de errores tipo memoria (*memory-style error-correcting-code organization*, ECC). Los sistemas de memoria hace tiempo que realizan la detección de errores utilizando los bits de paridad. Cada byte del sistema de memoria puede tener asociado un bit de paridad que registra si el número de bits del byte que valen uno es par



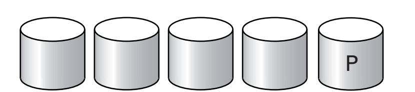
(a) RAID 0: Distribución no redundante



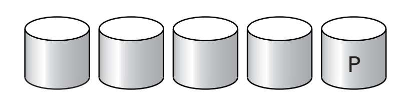
(b) RAID 1: Discos con imagen



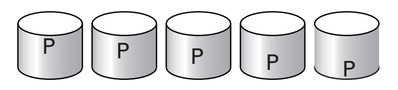
(c) RAID 2: Códigos de corrección de errores tipo memoria



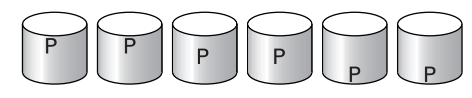
(d) RAID 3: Paridad con bits entrelazados



(e) RAID 4: Paridad con bloques entrelazados



(f) RAID 5: Paridad distribuida con bloques entrelazados



(g) RAID 6: Redundancia P + Q

FIGURA 11.4. Niveles de RAID.

(paridad = 0) o impar (paridad = 1). Si uno de los bits del byte se deteriora (un uno se transforma en cero o viceversa) la paridad del byte se modifica y, por tanto, no coincidirá con la paridad guardada. De manera parecida, si el bit de paridad guardado se deteriora no coincidirá con la paridad calculada. Por tanto, todos los errores de un bit son detectados por el sistema de memoria. Los esquemas de corrección de errores guardan dos o más bits adicionales y pueden reconstruir los datos si se deteriora un solo bit.

La idea de los códigos para la corrección de errores puede utilizarse directamente en los conjuntos de discos mediante la distribución de los bytes entre los discos. Por ejemplo, el primer bit de cada byte puede guardarse en el disco uno, el segundo en el disco dos, etcétera, hasta que se guarde el octavo bit en el disco ocho y los bits para la corrección de errores se guardan en los demás discos.

Este esquema se muestra gráficamente en la Figura 11.4. Los discos marcados como *P* guardan los bits para la corrección de errores. Si uno de los discos falla, los bits restantes del byte y los bits para la corrección de errores asociados pueden leerse de los demás discos y utilizarse para reconstruir los datos deteriorados. En la Figura 11.4c se muestra una disposición de tamaño cuatro; obsérvese que RAID de nivel 2 sólo necesita la sobrecarga de tres discos para cuatro discos de datos, a diferencia de RAID de nivel 1, que necesitaba la sobrecarga de cuatro discos.

- **RAID de nivel 3**, u organización de paridad con bits entrelazados, mejora respecto al nivel 2 destacando que, a diferencia de los sistemas de memoria, los controladores de disco pueden detectar si un sector se ha leído correctamente, por lo que se puede utilizar un solo bit de paridad para la corrección y la detección de los errores. La idea es la siguiente. Si uno de los sectores se deteriora, se sabe exactamente el sector que es y para cada bit del mismo se puede determinar si es un uno o un cero calculando la paridad de los bits correspondientes a partir de los sectores de los demás discos. Si la paridad de los bits restantes es igual que la paridad guardada, el bit ausente es un cero; en caso contrario, es un uno.

  RAID de nivel 3 es tan bueno como el nivel 2, pero resulta menos costoso en cuanto al número de discos adicionales (sólo tiene la sobrecarga de un disco), por lo que el nivel 2 no se utiliza en la práctica. Este esquema se muestra gráficamente en la Figura 11.4d.

  RAID de nivel 3 tiene dos ventajas respecto al nivel 1. Sólo se necesita un disco de paridad para varios discos normales, en comparación con un disco imagen por cada disco en el nivel 1, por lo que se reduce la sobrecarga de almacenamiento. Dado que los procesos de lectura y de escritura de un byte se extienden en varios discos, con la distribución de datos en *N* vías, la velocidad de transferencia es *N* veces tan rápida como con un solo disco. Por otro lado, RAID de nivel 3 permite un menor número de operaciones de E/S por segundo, dado que todos los discos tienen que participar en cada solicitud de E/S.

- **RAID de nivel 4**, u organización de paridad con bloques entrelazados, usa distribución de bloques, y además guarda un bloque de paridad en un disco aparte para los bloques correspondientes de los otros *N* discos. Este esquema se muestra gráficamente en la Figura 11.4e. Si uno de los discos falla puede utilizarse el bloque de paridad con los bloques correspondientes de los demás discos para restaurar los bloques del disco averiado.

  La lectura de un bloque sólo accede a un disco, lo que permite que los demás discos procesen otras solicitudes. Por tanto, la velocidad de transferencia de datos de cada acceso es menor, pero se pueden ejecutar en paralelo varios accesos de lectura, lo que produce una mayor velocidad global de E/S. Las velocidades de transferencia para los procesos de lectura de gran tamaño son elevadas, dado que se pueden leer todos los discos en paralelo; los procesos de escritura de gran tamaño también tienen velocidades de transferencia elevadas, dado que los datos y la paridad pueden escribirse en paralelo.

  Los procesos de escritura independientes de pequeño tamaño, por otro lado, no pueden realizarse en paralelo. La escritura de un bloque tiene que tener acceso al disco en el que se guarda ese bloque, así como al disco de paridad, dado que hay que actualizar el bloque de paridad. Además, hay que leer tanto el valor anterior del bloque de paridad como el del bloque que se escribe para calcular la nueva paridad. Por tanto, un solo proceso de escritura necesita cuatro accesos a disco: dos para leer los dos bloques antiguos y otros dos para escribir los dos nuevos.

- **RAID de nivel 5**, o paridad distribuida con bloques entrelazados, mejora respecto al nivel 4 dividiendo los datos y la paridad entre los $N + 1$ discos en vez de guardar los datos en *N* discos y la paridad en uno. En el nivel 5 todos los discos pueden participar en la atención a las solicitudes de lectura, a diferencia de RAID de nivel 4, en que el disco de paridad no puede participar, por lo que el nivel 5 aumenta el número total de solicitudes que pueden atenderse en una cantidad de tiempo dada. Para cada conjunto de *N* bloques lógicos, uno de los discos guarda la paridad y los otros *N* guardan los bloques.

  Esta configuración se muestra gráficamente en la Figura 11.4f, en la que las *P* están distribuidas entre todos los discos. Por ejemplo, con una disposición de cinco discos el bloque de paridad, marcado como Pk para los bloques lógicos $4k$, $4k+1$,

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

$4k+2$, $4k+3$ se guardan en el disco (k mod 5) + 1; los bloques correspondientes de los otros cuatro discos guardan los cuatro bloques de datos $4k$ a $4k+3$. La siguiente tabla muestra cómo se disponen los primeros veinte bloques, numerados de 0 a 19, y sus bloques de paridad. El patrón mostrado se repite para los siguientes bloques.

Obsérvese que un bloque de paridad no puede guardar la paridad de los bloques del mismo disco, dado que entonces un fallo del disco supondría la pérdida de datos además de la de la paridad y, por tanto, no sería recuperable. El nivel 5 incluye al nivel 4, dado que ofrece mejor rendimiento de lectura y de escritura por el mismo coste, por lo que el nivel 4 no se utiliza en la práctica.

- **RAID de nivel 6**, también denominado esquema de redundancia P+Q, es muy parecido a RAID de nivel 5 pero guarda información redundante adicional para protección contra fallos de disco múltiples. En lugar de utilizar la paridad se utilizan códigos para la corrección de errores como los de Reed-Solomon (véanse las notas bibliográficas). En el esquema mostrado en la Figura 11.4g se guardan dos bits de datos redundantes por cada cuatro bits de datos (en comparación con un bit de paridad del nivel 5) y el sistema puede tolerar dos fallos del disco.

Finalmente, se debe destacar que se han propuesto varias alternativas a los esquemas RAID básicos aquí descritos.

Algunos fabricantes usan su propia terminología para describir sus implementaciones RAID[2]. Sin embargo, la terminología que se ha presentado es la más ampliamente usada.

11.3.4. Elección del nivel RAID adecuado

Los factores a tener en cuenta al elegir un nivel RAID son:

- Costo económico extra de los requisitos de almacenamiento en disco.
- Requisitos de rendimiento en términos del número de operaciones E/S.
- Rendimiento cuando falla un disco.
- Rendimiento durante la reconstrucción (esto es, mientras los datos del disco averiado se reconstruyen en un nuevo disco).

Si un disco falla, el tiempo que se tarda en reconstruir los datos que contenía puede ser significativo, y variará con el nivel RAID utilizado. La reconstrucción resulta más sencilla para RAID de nivel 1, dado que los datos pueden copiarse de otro disco; para los otros niveles hay que tener acceso a todos los demás discos de la disposición para reconstruir los datos del disco averiado. El **rendimiento en la reconstrucción** de un sistema RAID puede ser un factor importante si se necesita un aporte continuo de datos, como ocurre en los sistemas de bases de datos de alto rendimiento. Además, dado que el tiempo de reconstrucción puede formar parte del tiempo de reparación, el rendimiento de la reconstrucción influye en el tiempo medio entre fallos.

RAID de nivel 0 se usa en aplicaciones de alto rendimiento donde la seguridad de los datos no es crítica. Dado que los niveles 2 y 4 de RAID se incluyen en los niveles 3 y 5 de RAID, la elección de los niveles RAID se limita a los niveles restantes. La distribución de bits (nivel 3) se usa raramente, dado que la distribución de bloques (nivel 5) da buenas velocidades de transferencia de datos para grandes transferencias. Para pequeñas transferencias, el tiempo de acceso a disco es el factor dominante, así que el beneficio de las lecturas paralelas disminuye. De hecho, el nivel 3 puede funcionar peor que el nivel 5 para una pequeña transferencia, ya que la transferencia sólo se completa cuando los sectores correspondientes en todos los discos se hayan encontrado; la latencia media de la disposición de discos se comporta de forma muy parecida a la latencia en el caso peor para un único disco, descartando los beneficios de las mayores velocidades de transferencia. El nivel 6 no se soporta actualmente en muchas implementaciones RAID, pero ofrece una mejor fiabilidad que el nivel 5 y se puede usar en aplicaciones donde la seguridad de datos es muy importante.

La elección entre RAID de nivel 1 y de nivel 5 es más difícil de tomar. RAID de nivel 1 es popular para las aplicaciones como el almacenamiento de archivos de registro histórico en un sistema de bases de datos, ya que ofrece el mejor rendimiento en escritura. RAID de nivel 5 tiene una menor sobrecarga de almacenamiento que el nivel 1, pero tiene una mayor sobrecarga en las escrituras. Para las aplicaciones donde los datos se leen frecuentemente y se escriben raramente, el nivel 5 es la elección adecuada.

Las capacidades de almacenamiento en disco han estado aumentando a una velocidad sobre el 50 por ciento al año durante muchos años, y el costo por byte

[2] Por ejemplo, algunos productos usan el nivel 1 de RAID para referirse a discos imagen sin distribución y el nivel 1 + 0 o 10 para los discos imagen con distribución. Esta distinción no es realmente necesaria, ya que la no distribución se puede ver como un caso especial de distribución, en concreto, la distribución sobre un único disco.

ha estado decreciendo a la misma velocidad. Como resultado, para muchas aplicaciones existentes de bases de datos con requisitos moderados de almacenamiento, el costo económico del almacenamiento extra en disco necesario para la creación de imágenes ha sido relativamente pequeño (sin embargo, el costo económico extra, sigue siendo un aspecto significativo para las aplicaciones de almacenamiento intensivo como el almacenamiento de datos de vídeo). Las velocidades de acceso se han mejorado a una velocidad mucho menor (cerca de un factor 3 durante 10 años), mientras que el número de operaciones E/S requeridas por segundo se han incrementado enormemente, particularmente en los servidores de aplicaciones Web.

El nivel 5 de RAID, que incrementa el número de operaciones E/S necesarias para escribir un único bloque lógico, sufre una penalización de tiempo significativa en términos del rendimiento en escritura. El nivel 1 de RAID es, por tanto, la elección adecuada para muchas aplicaciones con requisitos moderados de almacenamiento y altos requisitos de E/S.

Los diseñadores de sistemas RAID tienen también que hacer otras decisiones. Por ejemplo, la cantidad de discos que habrá en la disposición y los bits que debe proteger cada bit de paridad. Si hay más discos en la disposición, las velocidades de transferencia de datos son mayores pero el sistema sería más caro. Si hay más bits protegidos por cada bit de paridad, la sobrecarga de espacio debida a los bits de paridad es menor, pero hay más posibilidades de que falle un segundo disco antes de que el primer disco en averiarse esté reparado y que eso dé lugar a la pérdida de datos.

11.3.5. Aspectos hardware

Otro aspecto en la elección de implementaciones RAID se encuentra en el nivel hardware. RAID se puede implementar sin cambios en el nivel hardware modificando sólo el software. Tales implementaciones se conocen como **RAID software**. Sin embargo, hay beneficios significativos al construir hardware de propósito especial para dar soporte a RAID, que se describen a continuación; los sistemas con soporte hardware especial se denominan sistemas **RAID hardware**.

Las implementaciones RAID hardware pueden usar RAM no volátil para registrar las escrituras que es necesario ejecutar; en caso de fallo de corriente antes de que se complete una escritura, el sistema se restaura recuperando información acerca de las escrituras incompletas de la memoria RAM no volátil y entonces completa las escrituras. Sin el soporte hardware, se necesita trabajo extra para detectar los bloques que se hayan escrito parcialmente antes del fallo de corriente (véase el Ejercicio 11.4).

Algunas implementaciones RAID permiten el **intercambio en caliente**; esto es, los discos averiados se pueden eliminar y reemplazar por otros nuevos sin desconectar la corriente. El intercambio en caliente reduce el tiempo medio de reparación, ya que el cambio de un disco no debe esperar hasta que se pueda apagar el sistema. De hecho, muchos sistemas críticos actuales se ejecutan con una planificación 24 × 7; esto es, se ejecutan 24 horas al día y 7 días a la semana, sin proporcionar ningún momento para apagar el sistema y cambiar el disco averiado. Como resultado, el tiempo medio de reparación se reduce en gran medida, minimizando la posibilidad de pérdida de datos. El disco averiado se puede reemplazar en los ratos libres.

La fuente de alimentación, o el controlador de disco, o incluso la interconexión del sistema en un sistema RAID podría llegar a ser un punto de fallo que detendría el funcionamiento del sistema RAID. Para evitar esta posibilidad, las buenas implementaciones RAID tienen varias fuentes de alimentación (con baterías de respaldo que les permiten continuar funcionando aunque se corte la corriente). Tales sistemas RAID tienen varios controladores de disco y varias interconexiones para conectarlos con el sistema informático (o a la red de los sistemas informáticos). Así, el fallo de cualquier componente no detendrá el funcionamiento del sistema RAID.

11.3.6. Otras aplicaciones de RAID

Los conceptos de RAID se han generalizado a otros dispositivos de almacenamiento, incluyendo los conjuntos de cintas, e incluso a la transmisión de datos por sistemas de radio. Cuando se aplican a los conjuntos de cintas, las estructuras RAID pueden recuperar datos aunque se deteriore una de las cintas de la disposición. Cuando se aplican a la transmisión de datos, cada bloque de datos se divide en unidades menores y se transmite junto con una unidad de paridad; si por algún motivo no se recibe alguna de las unidades, se puede reconstruir a partir del resto.

11.4. ALMACENAMIENTO TERCIARIO

En un sistema de bases de datos de gran tamaño puede que parte de los datos tenga que residir en almacenamiento terciario. Los dos medios de almacenamiento terciario más frecuentes son los discos ópticos y las cintas magnéticas.

11.4.1. Discos ópticos

Los discos compactos son un medio popular de distribución de software, datos multimedia como el sonido y las imágenes, y otra información editada de manera

electrónica. Tienen una elevada capacidad de almacenamiento (640 megabytes) y resulta barato producir masivamente los discos.

Los discos de vídeo digital (DVD, Digital Video Disk) están reemplazando a los discos compactos en las aplicaciones que requieren grandes cantidades de datos. Los discos en formato DVD-5 pueden almacenar 4,7 gigabytes de datos (en una superficie de grabación), mientras que los discos en formato DVD-9 pueden almacenar 8,5 gigabytes de datos (en dos superficies de disco). La grabación en ambas caras de un disco ofrece incluso mayores capacidades; los formatos DVD-10 y DVD-18, que son las versiones de doble cara de DVD-5 y DVD-9, pueden almacenar respectivamente 9,4 y 17 gigabytes.

Las unidades CD y DVD presentan tiempos de búsqueda mucho mayores (100 milisegundos es un valor frecuente) que las unidades de discos magnéticos, debido a que el dispositivo de cabezas es más pesado. Las velocidades rotacionales son generalmente menores, aunque las unidades CD y DVD más rápidas tienen velocidades rotacionales alrededor de 3.000 revoluciones por minuto, que son comparables a las velocidades de los discos magnéticos de gama baja. Las velocidades rotacionales se correspondían inicialmente con las normas de los CD de sonido, y las velocidades de las unidades DVD se correspondían inicialmente con las normas de los DVD de vídeo, pero hoy en día se dispone de unidades que hacen girar los discos muchas veces la velocidad normal.

Las velocidades de transferencia de datos son algo menores que los discos magnéticos. Las unidades CD actuales leen entre 3 y 6 megabytes por segundo, y las unidades DVD actuales leen de 8 a 15 megabytes por segundo. Al igual que las unidades de discos magnéticos, los discos ópticos almacenan más datos en las pistas exteriores y menos en las interiores. La velocidad de transferencia de las unidades ópticas se caracteriza por $n\times$, que significa que la unidad soporta transferencias n veces la velocidad normal; las velocidades 50× para CD y 12× para DVD son comunes actualmente.

Las versiones de escritura única de los discos ópticos (CD-R y DVD-R) son populares para la distribución de datos y en particular para el almacenamiento de archivo de datos porque tienen una gran capacidad, un tiempo de vida mayor que los discos magnéticos y pueden retirarse de la unidad y almacenarse en un lugar remoto. Dado que no pueden sobrescribirse, se pueden usar para almacenar información que no debería cambiarse, como los rastros de auditoría. Las versiones de varias escrituras (CR-RW, DVD-RW y DVD-RAM) también se usan para archivo.

Los **cambiadores de discos** son dispositivos que guardan un gran número de discos ópticos (hasta varios cientos) y los cargan automáticamente bajo demanda en una de las pocas unidades (usualmente entre una y diez). La capacidad de almacenamiento agregada de tal sistema puede tener muchos terabytes. Cuando se accede a un disco, se carga mediante un brazo mecánico desde una estantería en la unidad (cualquier disco que estuviese previamente en la unidad se debe devolver a la estantería). El tiempo de carga y descarga suele ser del orden de segundos (mucho más lento que los tiempos de acceso a disco).

11.4.2. Cintas magnéticas

Aunque las cintas magnéticas son relativamente permanentes y pueden albergar grandes volúmenes de datos, resultan lentas en comparación con los discos magnéticos y ópticos. Aún más importante, la cinta magnética está limitada al acceso secuencial. Por tanto, resulta inadecuada para proporcionar el acceso aleatorio que cumple la mayor parte de los requisitos del almacenamiento secundario, aunque históricamente, antes del uso de los discos magnéticos, las cintas se usaron como medio de almacenamiento secundario.

Las cintas se utilizan principalmente para copias de seguridad, para el almacenamiento de la información poco utilizada y como medio sin conexión para transferir información de un sistema a otro. Las cintas también se usan para almacenar grandes volúmenes de datos, tales como datos vídeo o de imagen que, o no es necesario acceder rápidamente a ellos o son tan voluminosos que el almacenamiento en disco sería muy caro.

Una cinta se guarda en una bobina y se enrolla o desenrolla sobre una cabeza de lectura y escritura. El desplazamiento hasta el punto correcto de la cinta puede tardar minutos en vez de milisegundos; una vez en posición, sin embargo, las unidades de cinta pueden escribir los datos con densidades y velocidades que se aproximan a las de las unidades de disco. Las capacidades varían en función de la longitud y de la anchura de la cinta y de la densidad con la que la cabeza pueda leer y escribir. El mercado está actualmente dividido en una amplia variedad de formatos de cinta. Las capacidades actuales disponibles varían de unos pocos gigabytes (con el formato **DAT [Digital Audio Tape, cinta de audio digital**]), 10 a 40 gigabytes (con el formato **DLT [Digital Linear Tape, cinta lineal digital**]), 100 gigabytes y aún más (con el formato **Ultrium**), hasta 330 gigabytes (con los formatos de cinta de **exploración helicoidal de Ampex**). Las velocidades de transferencia de datos son del orden de algunos hasta decenas de megabytes por segundo.

Las unidades de cinta son muy fiables y los buenos sistemas de unidades de cinta realizan la lectura de los datos recién escritos para asegurar que se han registrado correctamente. Sin embargo, las cintas presentan límites en cuanto al número de veces que se pueden leer o escribir con fiabilidad.

Algunos formatos de cinta (como el formato **Accelis**) soportan velocidades de búsqueda mayores (del orden de decenas de segundos), lo cual es importante para las aplicaciones que necesitan un rápido acceso a muy grandes cantidades de datos, mayores de lo que

económicamente podría caber en una unidad de disco. La mayoría del resto de formatos de cinta proporcionan mayores capacidades al precio de un acceso más lento; estos formatos son ideales para la copia de seguridad de datos, donde las búsquedas rápidas no son importantes.

Los *cambiadores de cintas*, al igual que los cambiadores de discos ópticos, guardan gran número de cintas con unas cuantas unidades en las que se pueden montar las cintas; se utilizan para guardar grandes volúmenes de datos, que pueden llegar a varios terabytes (10^{12} bytes) con tiempos de acceso del orden de segundos hasta unos cuantos minutos. Las aplicaciones que necesitan estos enormes almacenamientos de datos incluyen los sistemas de imágenes que reúnen los datos de los satélites de teledetección, y grandes bibliotecas de vídeo para las cadenas de televisión.

11.5. ACCESO AL ALMACENAMIENTO

Las bases de datos se corresponden con cierto número de archivos diferentes que mantiene el sistema operativo subyacente. Estos archivos residen permanentemente en los discos, con copias de seguridad en cinta. Cada archivo se divide en unidades de almacenamiento de longitud constante denominadas **bloques**, que son las unidades de asignación de almacenamiento y de transferencia de datos. En el Apartado 11.6 se discutirán varias maneras de organizar los datos en archivos de manera lógica.

Cada bloque puede contener varios elementos de datos. El conjunto concreto de elementos de datos que contiene cada bloque viene determinado por la forma de organización física de los datos que se utilice (véase el Apartado 11.6). Se supondrá que ningún elemento de datos ocupa dos o más bloques. Esta suposición es realista para la mayor parte de las aplicaciones de procesamiento de datos, como el ejemplo bancario aquí propuesto.

Uno de los principales objetivos del sistema de bases de datos es minimizar el número de transferencias de bloques entre el disco y la memoria. Una manera de reducir el número de accesos al disco es mantener en la memoria principal todos los bloques que sea posible. El objetivo es maximizar la posibilidad de que, cuando se tenga acceso a un bloque, ya se encuentre en la memoria principal y, por tanto, no se necesite realizar un acceso al disco.

Dado que no resulta posible mantener en la memoria principal todos los bloques hay que gestionar la asignación del espacio disponible en la memoria principal para el almacenamiento de los mismos. La **memoria intermedia** (buffer) es la parte de la memoria principal disponible para el almacenamiento de las copias de los bloques del disco. Siempre se guarda en el disco una copia de cada bloque, pero esta copia puede ser una versión del bloque más antigua que la versión de la memoria intermedia. El subsistema responsable de la asignación del espacio de la memoria intermedia se denomina **gestor de la memoria intermedia**.

11.5.1. Gestor de la memoria intermedia

Los programas de un sistema de bases de datos formulan solicitudes (es decir, llamadas) al gestor de la memoria intermedia cuando necesitan un bloque del disco. Si el bloque ya se encuentra en la memoria intermedia se pasa al solicitante la dirección del bloque en la memoria principal. Si el bloque no se halla en la memoria intermedia, el gestor de la memoria intermedia asigna en primer lugar espacio al bloque en la memoria intermedia, descartando algún otro bloque si hace falta para hacer sitio para el nuevo bloque. Sólo se vuelve a escribir en el disco el bloque que se descarta si se modificó desde la última vez que se escribió en el disco. A continuación, el gestor de la memoria intermedia lee el bloque del disco y lo escribe en la memoria intermedia, y pasa la dirección del bloque en la memoria principal al solicitante. Las acciones internas del gestor de la memoria intermedia resultan transparentes para los programas que formulan solicitudes de bloques de disco.

Si se está familiarizado con los conceptos de los sistemas operativos se observará que el gestor de la memoria intermedia no parece ser más que un gestor de la memoria virtual, como los que se hallan en la mayor parte de los sistemas operativos. Una diferencia estriba en que el tamaño de la base de datos puede ser mucho mayor que el espacio de direcciones de hardware de la máquina, por lo que las direcciones de memoria no resultan suficientes para direccionar todos los bloques de memoria. Además, para dar un buen servicio al sistema de bases de datos el gestor de la memoria intermedia debe utilizar técnicas más complejas que los esquemas de gestión de la memoria virtual habituales:

- **Estrategia de sustitución.** Cuando no queda espacio libre en la memoria intermedia hay que eliminar un bloque de ésta antes de que se pueda escribir en él otro nuevo. Generalmente los sistemas operativos utilizan un esquema **menos recientemente utilizado** (Least Recently Used, LRU), en el que se vuelve a escribir en el disco y se elimina de la memoria intermedia el bloque al que se ha hecho referencia menos recientemente.
- **Bloques clavados.** Para que el sistema de bases de datos pueda recuperarse de las caídas (Capítulo 17) resulta necesario limitar las ocasiones en que se puede volver a escribir el bloque en el dis-

co. Se dice que un bloque al que no se le permite que se vuelva a escribir en el disco está **clavado**. Aunque muchos sistemas operativos no permiten trabajar con bloques clavados, esta prestación resulta esencial para la implementación de un sistema de bases de datos resistente a las caídas.

- **Salida forzada de los bloques.** Hay situaciones en las que resulta necesario volver a escribir el bloque en el disco, aunque no se necesite el espacio de memoria intermedia que ocupa. Este proceso de escritura se denomina **salida forzada** del bloque. Se verá el motivo de que se necesite la salida forzada en el Capítulo 17; para resumirlo, el contenido de la memoria principal y, por tanto, el de la memoria intermedia se pierde en las caídas, mientras que los datos del disco suelen sobrevivir a ellos.

11.5.2. Políticas para la sustitución de la memoria intermedia

El objetivo de las estrategias de sustitución de los bloques de la memoria intermedia es la minimización de los accesos al disco. En los programas de propósito general no resulta posible predecir con precisión los bloques a los que se hará referencia. Por tanto, los sistemas operativos utilizan la pauta anterior de las referencias a los bloques como forma de predecir las referencias futuras. La suposición que suele hacerse es que es probable que se vuelva a hacer referencia a los bloques a los que se ha hecho referencia recientemente. Por tanto, si hay que sustituir un bloque, se sustituye el bloque al que se ha hecho referencia menos recientemente. Este enfoque se denomina **esquema de sustitución de bloques LRU**.

LRU es un esquema de sustitución aceptable para los sistemas operativos. Sin embargo, los sistemas de bases de datos pueden predecir la pauta de las referencias futuras con más precisión que los sistemas operativos. Las peticiones del usuario al sistema de bases de datos comprenden varias etapas. El sistema de bases de datos suele poder determinar con antelación los bloques que se necesitarán examinando cada una de las etapas necesarias para llevar a cabo la operación solicitada por el usuario. Por tanto, a diferencia de los sistemas operativos, que deben confiar en el pasado para predecir el futuro, los sistemas de bases de datos pueden tener información concerniente al menos al futuro a corto plazo.

Para ilustrar la manera en que la información relativa al futuro acceso a los bloques permite mejorar la estrategia LRU considérese el procesamiento de la expresión del álgebra relacional

$$prestatario \bowtie cliente$$

Supóngase que la estrategia escogida para procesar esta solicitud viene dada por el programa en seudocódigo mostrado en la Figura 11.5 (se estudiarán otras estrategias en el Capítulo 13).

Supóngase que las dos relaciones de este ejemplo se guardan en archivos diferentes. En este ejemplo se puede ver que, una vez que se haya procesado la tupla de *prestatario*, no se vuelve a necesitar. Por tanto, una vez que se ha completado el procesamiento de un bloque completo de tuplas de *prestatario*, ese bloque ya no se necesita en la memoria principal, aunque se haya utilizado recientemente. Deben darse instrucciones al gestor de la memoria intermedia para liberar el espacio ocupado por el bloque de *prestatario* tan pronto como se haya procesado la última tupla. Esta estrategia de gestión de la memoria intermedia se denomina estrategia *de* **extracción inmediata**.

Considérense ahora los bloques que contienen las tuplas de *cliente*. Hay que examinar una vez cada bloque de las tuplas *cliente* por cada tupla de la relación *prestatario*. Cuando se completa el procesamiento del bloque *cliente* se sabe que no se tendrá nuevamente acceso a este hasta que se hayan procesado todos los demás bloques de *cliente*. Por tanto, el bloque de *cliente* al que se haya hecho referencia más recientemente será el último bloque al que se vuelva a referenciar, y el bloque de *cliente* al que se haya hecho referencia menos recientemente será el bloque al que se vuelva a hacer referencia a continuación. Este conjunto de suposiciones es el reverso exacto del que forma la base de la estrategia LRU. En realidad, la estrategia óptima para la sustitución de bloques es la **estrategia más recientemente**

```
for each tupla p de prestatario do
    for each tupla c de cliente do
        if p[nombre-cliente] = c[nombre-cliente]
        then begin
            sea x una tupla definida de la manera siguiente:
            x[nombre-cliente] := p[nombre-cliente]
            x[número-préstamo] := p[número-préstamo]
            x[calle-cliente] := c[calle-cliente]
            x[ciudad-cliente] := c[ciudad-cliente]
            incluir la tupla x como parte del resultado de prestatario ⋈ cliente
        end

    end
end
```

FIGURA 11.5. Procedimiento para calcular la reunión.

utilizada (**Most Recently Used**, **MRU**). Si hay que eliminar de la memoria intermedia un bloque de *cliente*, la estrategia MRU escoge el bloque utilizado más recientemente.

Para que la estrategia MRU funcione correctamente en el ejemplo propuesto, el sistema debe clavar el bloque de *cliente* que se esté procesando. Después de que se haya procesado la última tupla de *cliente* el bloque se desclava y se transforma en el bloque utilizado más recientemente.

Además de utilizar la información que pueda tener el sistema respecto de la solicitud que se esté procesando, el gestor de la memoria intermedia puede utilizar información estadística concerniente a la probabilidad de que una solicitud haga referencia a una relación concreta. Por ejemplo, el diccionario de datos (como se verá en detalle en el Apartado 11.8) que guarda el esquema lógico de las relaciones y su información del almacenamiento físico es una de las partes de la base de datos a la que se tiene acceso con mayor frecuencia. Por tanto, el gestor de la memoria intermedia debe intentar no eliminar de la memoria principal los bloques del diccionario de datos a menos que se vea obligado a hacerlo por otros factores. En el Capítulo 12 se discuten los índices de los archivos. Dado que puede que se tenga acceso más frecuentemente al índice de un archivo que al propio archivo, el gestor de la memoria intermedia no deberá, en general, eliminar los bloques del índice de la memoria principal si se dispone de alternativas.

La estrategia ideal para la sustitución de bloques necesita información sobre las operaciones de la bases de datos (las que se estén realizando y las que se realizarán en el futuro). No se conoce ninguna estrategia aislada que responda bien a todas las situaciones posibles. En realidad, un número sorprendentemente grande de bases de datos utilizan LRU a pesar de los defectos de esa estrategia. Los ejercicios exploran estrategias alternativas.

La estrategia utilizada por el gestor de la memoria intermedia para la sustitución de los bloques se ve influida por factores distintos del momento en que se volverá a hacer referencia al bloque. Si el sistema está procesando de manera concurrente las solicitudes de varios usuarios, puede que el subsistema para el control de la concurrencia (Capítulo 16) tenga que posponer ciertas solicitudes para asegurar la conservación de la consistencia de la base de datos. Si se proporciona al gestor de la memoria intermedia información del subsistema de control de la concurrencia que indique las solicitudes que se posponen, puede utilizar esta información para modificar su estrategia de sustitución de los bloques. De manera específica, los bloques que necesiten las solicitudes activas (no pospuestas) pueden retenerse en la memoria intermedia a expensas de los bloques que necesiten las solicitudes pospuestas.

El subsistema para la recuperación de caídas (Capítulo 17) impone restricciones estrictas a la sustitución de los bloques. Si se ha modificado un bloque, no se permite al gestor de la memoria intermedia volver a copiar la versión nueva del bloque de la memoria intermedia al disco, dado que eso destruiría la versión anterior. Por el contrario, el gestor de bloques debe solicitar permiso del subsistema para la recuperación de averías antes de escribir los bloques. Puede que el subsistema para la recuperación de averías exija que se fuerce la salida de otros bloques antes de conceder autorización al gestor de la memoria intermedia para escribir el bloque solicitado. En el Capítulo 17 se define con precisión la interacción entre el gestor de la memoria intermedia y el subsistema para la recuperación de averías.

11.6. ORGANIZACIÓN DE LOS ARCHIVOS

Los **archivos** se organizan lógicamente como secuencias de registros. Estos registros se corresponden con los bloques del disco. Los archivos se proporcionan como un instrumento fundamental de los sistemas operativos, por lo que se supondrá la existencia de un **sistema de archivos** subyacente. Hay que tomar en consideración diversas maneras de representar los modelos lógicos de datos en términos de archivos.

Aunque los bloques son de un tamaño fijo determinado por las propiedades físicas del disco y por el sistema operativo, los tamaños de los registros varían. En las bases de datos relacionales las tuplas de las diferentes relaciones suelen ser de tamaños distintos.

Un enfoque de la correspondencia entre la base de datos y los archivos es utilizar varios y guardar los registros de cada una de las diferentes longitudes fijas existentes en cada uno de esos archivos. Los archivos con registros de longitud fija son más sencillos de implementar que los archivos con registros de longitud variable. Muchas de las técnicas utilizadas para los primeros pueden aplicarse al caso de longitud variable. Por tanto, se comienza considerando un archivo con registros de longitud fija.

11.6.1. Registros de longitud fija

A manera de ejemplo, considérese un archivo con registros de *cuentas* de la base de datos bancaria. Cada registro de este archivo se define de la manera siguiente:

type *depósito* = **record**
número-cuenta: char(10);
nombre-sucursal: char (22);
saldo: real;
end

registro 0	C-102	Navacerrada	400
registro 1	C-305	Collado Mediano	350
registro 2	C-215	Becerril	700
registro 3	C-101	Centro	500
registro 4	C-222	Moralzarzal	700
registro 5	C-201	Navacerrada	900
registro 6	C-217	Galapagar	750
registro 7	C-110	Centro	600
registro 8	C-218	Navacerrada	700

FIGURA 11.6. Archivo que contiene los registros de *cuenta*.

registro 0	C-102	Navacerrada	400
registro 1	C-305	Collado Mediano	350
registro 8	C-218	Navacerrada	700
registro 3	C-101	Centro	500
registro 4	C-222	Moralzarzal	700
registro 5	C-201	Navacerrada	900
registro 6	C-217	Galapagar	750
registro 7	C-110	Centro	600

FIGURA 11.8. El archivo de la Figura 11.6 con el registro 2 borrado y el último registro desplazado.

Si se supone que cada carácter ocupa un byte y que un valor de tipo real ocupa ocho bytes, el registro de *cuenta* tiene cuarenta bytes de longitud. Un enfoque sencillo es utilizar los primeros cuarenta bytes para el primer registro, los cuarenta bytes siguientes para el segundo registro, etcétera (Figura 11.6). Sin embargo, hay dos problemas con este enfoque sencillo:

1. Resulta difícil borrar un registro de esta estructura. Se debe rellenar el espacio ocupado por el registro que hay que borrar con algún otro registro del archivo o tener algún medio de marcar los registros borrados para que puedan pasarse por alto.
2. A menos que el tamaño de los bloques sea un múltiplo de cuarenta (lo que resulta improbable) algunos de los registros se saltarán los límites de los bloques. Es decir, parte del registro se guardará en un bloque y parte en otro. Harán falta, por tanto, dos accesos a bloques para leer o escribir ese tipo de registros.

Cuando se borra un registro se puede desplazar el registro situado a continuación al espacio ocupado anteriormente por el registro borrado y hacer lo mismo con los demás registros hasta que todos los registros situados a continuación del borrado se hayan desplazado hacia delante (Figura 11.7). Un enfoque de este tipo necesita desplazar gran número de registros. Resultaría más sencillo desplazar simplemente el último registro del archivo al espacio ocupado por el registro borrado (Figura 11.8).

No resulta deseable desplazar los registros para ocupar el espacio liberado por los registros borrados, dado que se necesitan accesos adicionales a los bloques. Dado que las inserciones tienden a ser más frecuentes que los borrados, sí resulta aceptable dejar libre el espacio ocupado por los registros borrados y esperar a una inserción posterior antes de volver a utilizar ese espacio. No basta con una simple marca en el registro borrado, dado que resulta difícil el espacio disponible mientras se realiza una inserción. Por tanto, hay que introducir una estructura adicional.

Al comienzo del archivo se asigna cierto número de bytes como **cabecera del archivo**. La cabecera contendrá gran variedad de información sobre el archivo. Por ahora, todo lo que hace falta guardar ahí es la dirección del primer registro cuyo contenido se haya borrado. Se utiliza este primer registro para guardar la dirección del segundo registro disponible, y así sucesivamente. De manera intuitiva se pueden considerar estas direcciones guardadas como *punteros*, dado que indican la posición de un registro. Los registros borrados, por tanto, forman una lista enlazada a la que se suele denominar **lista libre**. En la Figura 11.9 se muestra el archivo de la Figura 11.6 después de haberse borrado los registros 1, 4 y 6.

Al insertar un registro nuevo se utiliza el registro indicado por la cabecera. Se cambia el puntero de la cabecera para que apunte al siguiente registro disponible. Si no hay espacio disponible, se añade el nuevo registro al final del archivo.

La inserción y el borrado de archivos con registros de longitud fija son sencillas de implementar, dado que el

registro 0	C-102	Navacerrada	400
registro 1	C-305	Collado Mediano	350
registro 3	C-101	Centro	500
registro 4	C-222	Moralzarzal	700
registro 5	C-201	Navacerrada	900
registro 6	C-217	Galapagar	750
registro 7	C-110	Centro	600
registro 8	C-218	Navacerrada	700

FIGURA 11.7. El archivo de la Figura 11.6 con el registro 2 borrado y todos los registros desplazados.

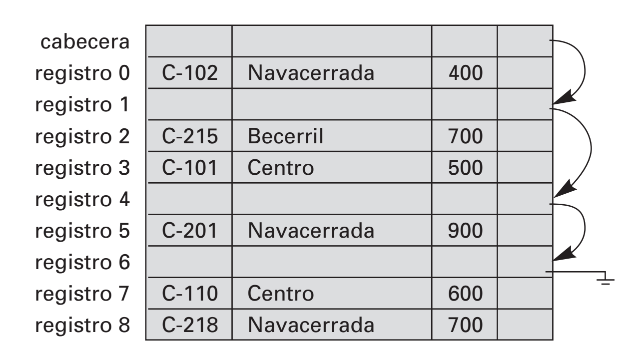


FIGURA 11.9. El archivo de la Figura 11.6 después del borrado de los registros 1, 4 y 6.

espacio que deja libre el registro borrado es exactamente el mismo que se necesita para insertar otro registro. Si se permiten en un archivo registros de longitud variable esta coincidencia no se mantiene. Puede que el registro insertado no quepa en el espacio liberado por el registro borrado o puede que sólo llene parte del mismo.

11.6.2. Registros de longitud variable

Los registros de longitud variable surgen de varias maneras en los sistemas de bases de datos:

- Almacenamiento de varios tipos de registros en un mismo archivo
- Tipos de registro que permiten longitudes variables para uno o varios de los campos
- Tipos de registro que permiten campos repetidos

Existen diferentes técnicas para implementar los registros de longitud variable. Con fines ilustrativos se utilizará un ejemplo para mostrar las diversas técnicas de implementación. Se tomará en consideración una representación diferente de la información de *cuenta* guardada en el archivo de la Figura 11.6, en la que se utiliza un registro de longitud variable para el nombre de cada sucursal y para toda la información de las cuentas de cada sucursal. El formato del registro es

type *lista-cuentas* = **record**
nombre-sucursal : char (22);
información-cuenta : **array** [1 .. ∞] **of record**;
número-cuenta : char(10);
saldo : real;
end
end

Se define *información-cuenta* como un *array* con un número arbitrario de elementos, por lo que no hay ningún límite para el tamaño que pueden tener los registros (hasta el tamaño del disco, ¡por supuesto!).

11.6.2.1. Representación en cadenas de bytes

Un método sencillo de implementar los registros de longitud variable es adjuntar un símbolo especial de *fin-de-registro* (⊥) al final de cada registro. Así se puede guardar cada registro como una cadena de bytes consecutivos. En la Figura 11.10 se muestra una organización de este tipo que representa el archivo con registros de longitud fija de la Figura 11.6 utilizando registros de longitud variable. Una versión alternativa de la representación en cadenas de bytes guarda la longitud del registro al comienzo de cada registro en lugar de utilizar símbolos de final de registro.

La representación en cadenas de bytes tal y como se ha discutido presenta varios inconvenientes:

- No resulta sencillo volver a utilizar el espacio ocupado anteriormente por un registro borrado. Aunque existen técnicas para gestionar la inserción y el borrado de registros, generan gran número de fragmentos pequeños de almacenamiento de disco desaprovechados.
- Por lo general no queda espacio para el aumento del tamaño de los registros. Si un registro de longitud variable aumenta de tamaño hay que desplazarlo (el movimiento resulta costoso si el registro está almacenado en otro lugar de la base de datos; por ejemplo, en los índices o en otros registros), ya que los punteros se deben localizar y actualizar.

Por tanto, no se suele utilizar la representación sencilla en cadenas de bytes tal y como aquí se ha descrito para implementar registros de longitud variable. Sin embargo, una forma modificada de la representación en cadenas de bytes, denominada estructura de páginas con ranuras, se utiliza normalmente para organizar los registros *dentro de* cada bloque.

La **estructura de páginas con ranuras** se muestra en la Figura 11.11. Hay una cabecera al principio de cada bloque que contiene la información siguiente:

1. El número de elementos del registro de la cabecera
2. El final del espacio vacío del bloque
3. Un *array* cuyas entradas contienen la ubicación y el tamaño de cada registro

Los registros reales se ubican *de manera contigua* en el bloque, empezando desde el final del mismo. El espacio libre dentro del bloque es contiguo, entre la última entrada del *array* de la cabecera y el primer registro. Si se inserta un registro se le asigna espacio al final del espacio libre y se añade a la cabecera una entrada que contiene su tamaño y su ubicación.

0	Navacerrada	C-102	400	C-201	900	C-218	700	⊥
1	Collado Mediano	C-305	350	⊥				
2	Becerril	C-215	700	⊥				
3	Centro	C-101	500	C-110	600	⊥		
4	Moralzarzal	C-222	700	⊥				
5	Galapagar	C-217	750	⊥				

FIGURA 11.10. Representación en cadenas de bytes de los registros de longitud variable.

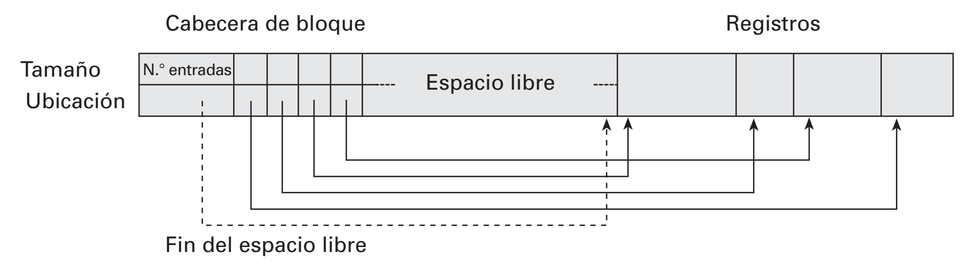


FIGURA 11.11. Estructura de páginas con ranuras.

Si se borra un registro se libera el espacio que ocupa y se da el valor de «borrada» a su entrada (por ejemplo, se le da a su tamaño el valor de –1). Además, se desplazan los registros de bloque situados antes del registro borrado, de modo que se ocupe el espacio libre creado por el borrado y todo el espacio libre vuelve a hallarse entre la última entrada del *array* de la cabecera y el primer registro. También se actualiza de manera adecuada el puntero de final del espacio libre de la cabecera. Se puede aumentar o disminuir el tamaño de los registros utilizando técnicas parecidas, siempre y cuando quede espacio en el bloque. El coste de trasladar los registros no es demasiado elevado, dado que el tamaño del bloque es limitado: un valor típico es cuatro kilobytes.

La estructura de páginas con ranuras necesita que no haya punteros que apunten directamente a los registros. Por el contrario, los punteros deben apuntar a la entrada de la cabecera que contiene la ubicación verdadera del registro. Este nivel de indirección permite a los registros desplazarse para evitar la fragmentación del espacio dentro del bloque al tiempo que permite los punteros indirectos a los registros.

11.6.2.2. Representación de longitud fija

Otra manera de implementar eficientemente los registros de longitud variable en un sistema de archivos es utilizar uno o varios registros de longitud fija para representar cada registro de longitud variable.

Hay dos técnicas para hacer esto:

1. **Espacio reservado.** Si hay una longitud de registro máxima que no se supera nunca, se pueden utilizar registros de longitud fija de esa longitud. El espacio no utilizado (por los registros más cortos que el espacio máximo) se rellena con un símbolo especial de valor nulo o de final de registro.
2. **Representación con listas.** El registro de longitud variable se representa mediante una lista de registros de longitud fija, enlazada mediante punteros.

Si se escoge aplicar el método del espacio reservado al ejemplo de las cuentas bancarias hay que seleccionar una longitud de registro máxima. En la Figura 11.12 se muestra el modo en que se representaría el archivo si se permitiera un máximo de tres cuentas por sucursal. Los registros de este archivo son del tipo *lista-cuentas*, pero el *array* contiene exactamente tres elementos. Las sucursales con menos de tres cuentas (por ejemplo, Collado Mediano) tienen registros con campos con valores nulos. En la Figura 11.12 se utiliza el símbolo ⊥ para representar esta situación. En la práctica se utiliza un valor concreto que no pueda representar nunca un dato real (por ejemplo, un «número de cuenta» negativo o un nombre que comience por un «*»).

El método del espacio reservado resulta útil cuando la mayor parte de los registros son de una longitud cercana a la máxima. En caso contrario se puede desperdiciar una cantidad de espacio significativa. En el ejemplo bancario puede que algunas sucursales tengan muchas más cuentas que otras. Esta situación lleva a considerar el uso del método de las listas enlazadas. Para representar el archivo utilizando el método de los punteros se añade un campo puntero igual que se hizo en la Figura 11.9. La estructura resultante se muestra en la Figura 11.13.

Las estructuras de archivo de las Figuras 11.9 y 11.13 son idénticas, salvo que en la Figura 11.9 sólo se utilizaron los punteros para enlazar los registros borrados, mientras que en la Figura 11.13 se enlazan todos los registros pertenecientes a la misma sucursal.

0	Navacerrada	C-102	400	C-201	900	C-218	700
1	Collado Mediano	C-305	350	⊥	⊥	⊥	⊥
2	Becerril	C-215	700	⊥	⊥	⊥	⊥
3	Centro	C-101	500	C-110	600	⊥	⊥
4	Moralzarzal	C-222	700	⊥	⊥	⊥	⊥
5	Galapagar	C-217	750	⊥	⊥	⊥	⊥

FIGURA 11.12. El archivo de la Figura 11.10 utilizando el método del espacio reservado.

0	Navacerrada	C-102	80.000
1	Collado Mediano	C-305	70.000
2	Becerril	C-215	140.000
3	Centro	C-101	100.000
4	Moralzarzal	C-222	140.000
5		C-201	180.000
6	Galapagar	C-217	150.000
7		C-110	120.000
8		C-218	140.000

FIGURA 11.13. El archivo de la Figura 11.10 utilizando el método de las listas enlazadas.

Un inconveniente de la estructura de la Figura 11.13 es que se desperdicia espacio en todos los registros excepto en el primero de la serie. El primer registro debe tener el valor *nombre-sucursal*, pero los registros siguientes no necesitan tenerlo. No obstante, hay que incluir en todos los registros un campo para *nombre-sucursal*, o los registros no serán de longitud constante. El espacio desperdiciado es significativo, dado que se espera en la práctica que cada sucursal tenga un gran número de cuentas. Para resolver este problema se permiten en el archivo dos tipos de bloques:

1. **Bloque ancla**, que contiene el primer registro de cada cadena.
2. **Bloque de desbordamiento**, que contiene los registros que no son los primeros de sus cadenas.

Por tanto, todos los registros *del interior de cada bloque* tienen la misma longitud, aunque no todos los registros del archivo tengan la misma longitud. En la Figura 11.14 se muestra esta estructura de archivos.

bloque ancla

Navacerrada	C-102	400	
Collado Mediano	C-305	350	
Becerril	C-215	700	
Centro	C-101	500	
Moralzarzal	C-222	700	
Galapagar	C-217	750	

bloque de desbordamiento

C-201	900	
C-218	700	
C-110	600	

FIGURA 11.14. Estructuras de bloque ancla y de bloque de desbordamiento.

11.7. ORGANIZACIÓN DE LOS REGISTROS EN ARCHIVOS

Hasta ahora se ha estudiado la manera en que se representan los registros en la estructura de los archivos. Un conjunto de registros constituye un ejemplo de esta relación. Dado un conjunto de registros, la pregunta siguiente es la manera de organizarlos en archivos. A continuación se indican varias de las maneras de organizar los registros en archivos:

- **Organización de archivos en montículo.** En esta organización se puede colocar cualquier registro en cualquier parte del archivo en que haya espacio suficiente. No hay ninguna ordenación de los registros. Generalmente sólo hay un archivo por cada relación.
- **Organización de archivos secuenciales.** En esta organización los registros se guardan en orden secuencial, basado en el valor de la clave de búsqueda de cada registro. La implementación de esta organización se describe en el Apartado 11.7.1.
- **Organización asociativa (*hash*) de archivos.** En esta organización se calcula una función de asociación (*hash*) de algún atributo de cada registro. El resultado de la función de asociación especifica el bloque del archivo en que se deberá colocar el registro. Esta organización se describe en el Capítulo 12; está estrechamente relacionada con las estructuras para la creación de índices descritas en dicho capítulo.

Generalmente se usa un archivo separado para almacenar los registros de cada relación. Sin embargo, en una **organización de archivos en agrupaciones** se pueden guardar en el mismo archivo registros de relaciones diferentes; además, los registros relacionados de las diferentes relaciones se guardan en el mismo bloque, por lo que cada operación de E/S afecta a registros relacionados de todas esas relaciones. Por ejemplo, los registros de las dos relaciones se pueden considerar relacionados si casan en una reunión de las dos relaciones. Esta organización se describe en el Apartado 11.7.2.

11.7.1. Organización de archivos secuenciales

Los **archivos secuenciales** están diseñados para el procesamiento eficiente de los registros de acuerdo con un orden basado en una clave de búsqueda. Una **clave de búsqueda** es cualquier atributo o conjunto de atributos; no tiene por qué ser una clave primaria, ni siquiera una superclave. Para permitir la recuperación rápida de los registros según el orden de la clave de búsqueda, los registros se vinculan mediante punteros. El puntero de cada registro apunta al siguiente registro según el orden indicado por la clave de búsqueda. Además, para minimizar el número de accesos a los bloques en el procesamiento de los archivos secuenciales, los registros se guardan físicamente de acuerdo con el orden indicado

por la clave de búsqueda, o en un orden tan cercano a éste como sea posible.

En la Figura 11.15 se muestra un archivo secuencial de registros de *cuenta* tomado del ejemplo bancario propuesto. En ese ejemplo los registros se guardan de acuerdo con el orden de la clave de búsqueda, utilizando como tal *nombre-sucursal*.

La organización secuencial de archivos permite que los registros se lean de forma ordenada, lo que puede ser útil para la visualización, así como para ciertos algoritmos de procesamiento de consultas que se estudiarán en el Capítulo 13.

Sin embargo, resulta difícil mantener el orden físico secuencial cuando se insertan y borran registros, dado que resulta costoso desplazar muchos registros como consecuencia de una sola inserción o borrado. Se puede gestionar el borrado utilizando cadenas de punteros, como ya se ha visto anteriormente. Para la inserción se aplican las reglas siguientes:

1. Localizar el registro del archivo que precede al registro que se va a insertar en el orden de la clave de búsqueda.
2. Si existe algún registro vacío (es decir, un espacio que haya quedado libre después de un borrado) dentro del mismo bloque que ese registro, el registro nuevo se insertará ahí. En caso contrario el nuevo registro se insertará en un *bloque de desbordamiento*. En cualquier caso, hay que ajustar los punteros para vincular los registros según el orden de la clave de búsqueda.

En la Figura 11.16 se muestra el archivo de la Figura 11.15 después de la inserción del registro (C-888, Leganés, 800). La estructura de la Figura 11.16 permite la inserción rápida de nuevos registros, pero obliga a las aplicaciones de procesamiento de archivos secuenciales a procesar los registros en un orden que no coincide con su orden físico.

Si hay que guardar un número relativamente pequeño de registros en los bloques de desbordamiento, este enfoque funciona bien. Finalmente, sin embargo, la correspondencia entre el orden de la clave de búsqueda y el orden físico puede perderse totalmente, en cuyo caso el procesamiento secuencial será significativamente menos eficiente. Llegados a este punto se debe **reorganizar** el archivo de modo que vuelva a estar físicamente en orden secuencial. Estas reorganizaciones resultan costosas y deben realizarse en momentos en los que la carga del sistema sea baja. La frecuencia con la que se necesitan las reorganizaciones depende de la frecuencia de inserción de registros nuevos. En el caso extremo en que las inserciones tengan lugar raramente, siempre resultará posible mantener el archivo en el orden físico correcto. En ese caso no es necesario el campo puntero mostrado en la Figura 11.15.

C-215	Becerril	700	
C-101	Centro	500	
C-110	Centro	600	
C-305	Collado Mediano	350	
C-217	Galapagar	750	
C-222	Moralzarzal	700	
C-102	Navacerrada	400	
C-201	Navacerrada	900	
C-218	Navacerrada	700	

FIGURA 11.15. Archivo secuencial para los registros de *cuenta*.

C-215	Becerril	700	
C-101	Centro	500	
C-110	Centro	600	
C-305	Collado Mediano	350	
C-217	Galapagar	750	
C-222	Moralzarzal	700	
C-102	Navacerrada	400	
C-201	Navacerrada	900	
C-218	Navacerrada	700	
C-888	Leganés	800	

FIGURA 11.16. El archivo secuencial después de una inserción.

11.7.2. Organización de archivos en agrupaciones

Muchos sistemas de bases de datos relacionales guardan cada relación en un archivo diferente de modo que puedan aprovechar completamente el sistema de archivos que forma parte del sistema operativo. Generalmente las tuplas de cada relación pueden representarse como registros de longitud fija. Por tanto, las relaciones pueden hacerse corresponder con una estructura de archivos sencilla. Esta implementación sencilla de los sistemas de bases de datos relacionales resulta adecuada para los sistemas de bases de datos diseñados para computadoras personales. En estos sistemas el tamaño de la base de datos es pequeño, por lo que se obtiene poco provecho de una estructura de archivos avanzada. Además, en algunas computadoras personales el pequeño tamaño global del código objeto del sistema de bases de datos resulta fundamental. Una estructura de archivos sencilla reduce la cantidad de código necesaria para implementar el sistema.

Este enfoque sencillo de la implementación de bases de datos relacionales resulta menos satisfactorio a medida que aumenta el tamaño de la base de datos. Ya se ha visto que se pueden obtener ventajas en el rendimiento mediante la asignación esmerada de los registros a los bloques y de la organización cuidadosa de los propios bloques. Por tanto, resulta evidente que puede resultar beneficiosa una estructura de archivos más compleja, aunque se mantenga la estrategia de guardar cada relación en un archivo diferente.

nombre-cliente	*número-cuenta*
López	C-102
López	C-220
López	C-503
Abril	C-305

FIGURA 11.17. La relación *impositor*.

López	Mayor	Arganzuela
López	C-102	
López	C-220	
López	C-503	
Abril	Preciados	Valsaín
Abril	C-305	

FIGURA 11.19. Estructura de archivo en agrupaciones.

Sin embargo, muchos sistemas de bases de datos de gran tamaño no utilizan directamente el sistema operativo subyacente para la gestión de archivos. Por el contrario, se asigna al sistema de bases de datos un archivo de gran tamaño del sistema operativo. En este archivo se guardan todas las relaciones y se confía la gestión de este archivo al sistema de bases de datos. Para ver la ventaja de guardar muchas relaciones en un solo archivo considérese la siguiente consulta SQL de la base de datos bancaria:

select *número-cuenta*, *nombre-cliente*, *calle-cliente*, *ciudad-cliente*
from *impositor*, *cliente*
where *impositor.nombre-cliente* = *cliente.nombre-cliente*

Esta consulta calcula una reunión de las relaciones *impositor* y *cliente*. Por tanto, por cada tupla *impositor* el sistema debe encontrar las tuplas *cliente* con el mismo valor de *nombre-cliente*. En teoría, estos registros se podrán encontrar con la ayuda de los *índices*, que se discutirán en el Capítulo 12. Independientemente de la manera en que se encuentren estos registros hay que transferirlos desde el disco a la memoria principal. En el peor de los casos cada registro se hallará en un bloque diferente, lo que obligará a efectuar un proceso de lectura de bloque por cada registro necesario para la consulta.

Como ejemplo concreto, considérense las relaciones *impositor* y *cliente* de las Figuras 11.17 y 11.18, respectivamente. En la Figura 11.19 se muestra una estructura de archivo diseñada para la ejecución eficiente de las consultas que implican *impositor* ⋈ *cliente*. Las tuplas *impositor* para cada *nombre-cliente* se guardan cerca de la tupla *cliente* para el *nombre-cliente* correspondiente. Esta estructura mezcla las tuplas de dos relaciones pero permite el procesamiento eficaz de la reunión. Cuando se lee una tupla de la relación *cliente* se copia del disco a la memoria principal todo el bloque que contiene esa tupla. Dado que las tuplas correspondientes de *impositor* se guardan en el disco cerca de la tupla *cliente*, el bloque que contiene la tupla *cliente* también contiene las tuplas de la relación *impositor* necesarias para procesar la consulta. Si un cliente tiene tantas cuentas que los registros de *impositor* no caben en un solo bloque, los registros restantes aparecerán en bloques cercanos.

Una **organización de archivos en agrupaciones** es una organización de archivos, como la mostrada en la Figura 11.19 que almacena registros relacionados de dos o más relaciones en cada bloque. Esta organización permite leer muchos de los registros que satisfacen la condición de reunión utilizando un solo proceso de lectura de bloques. Por tanto, se puede procesar esta consulta concreta de manera más eficiente.

Este uso de la agrupación ha mejorado el procesamiento de una reunión particular (*impositor* ⋈ *cliente*) pero ha producido el retardo del procesamiento de otros tipos de consulta. Por ejemplo,

select *
from *cliente*

necesita más accesos a los bloques que en el esquema en el que se guardaba cada relación en un archivo diferente. En lugar de que aparezcan varios registros de *cliente* en un mismo bloque, cada registro se halla en un bloque diferente. En realidad, hallar todos los registros de *cliente* no resulta posible sin alguna estructura adicional. Para encontrar todas las tuplas de la relación *cliente* en la estructura de la Figura 11.19 hay que vincular todos los registros de esa relación utilizando punteros, tal y como se muestra en la Figura 11.20.

La determinación del momento de utilizar la agrupación depende de los tipos de consulta que el diseñador de la base de datos considere más frecuentes. El uso cuidadoso de la agrupación puede producir ganancias de rendimiento significativas en el procesamiento de consultas.

nombre-cliente	*calle-cliente*	*ciudad-cliente*
López	Principal	Arganzuela
Abril	Preciados	Valsaín

FIGURA 11.18. La relación *cliente*.

López	Mayor	Arganzuela	
López	C-102		
López	C-220		
López	C-503		
Abril	Preciados	Valsaín	
Abril	C-305		

FIGURA 11.20. Estructura de archivo con agrupaciones con cadenas de punteros.

11.8. ALMACENAMIENTO CON DICCIONARIOS DE DATOS

Hasta ahora sólo se ha considerado la representación de las propias relaciones. Un sistema de bases de datos relacionales necesita tener datos *sobre* las relaciones, como el esquema de las mismas. Esta información se denomina **diccionario de datos** o **catálogo del sistema**. Entre los tipos de información que debe guardar el sistema figuran los siguientes:

- Los nombres de las relaciones
- Los nombres de los atributos de cada relación
- Los dominios y las longitudes de los atributos
- Los nombres de las vistas definidas en la base de datos y las definiciones de esas vistas
- Las restricciones de integridad (por ejemplo, las restricciones de las claves)

Además, muchos sistemas guardan los datos siguientes de los usuarios del sistema:

- Los nombres de los usuarios autorizados
- La información de las cuentas de usuarios
- Contraseñas u otra información usada para autenticar a los usuarios

Además, se puede guardar información estadística y descriptiva sobre estos asuntos:

- Número de tuplas de cada relación
- Método de almacenamiento utilizado para cada relación (por ejemplo, con agrupaciones o sin agrupaciones)

El diccionario de datos puede también anotar la organización del almacenamiento (secuencial, asociativa o con montículos) de las relaciones y la ubicación donde se almacena cada relación:

- Si las relaciones se almacenan en archivos del sistema operativo, el diccionario no podría anotar los nombres de los archivos que almacenan cada relación.
- Si la base de datos almacena todas las relaciones en un único archivo, el diccionario puede anotar los bloques que almacenan los registros de cada relación en una estructura de datos como una lista enlazada.

En el Capítulo 12, en el que se estudian los índices, se verá que hace falta guardar información sobre cada índice de cada una de las relaciones:

- El nombre del índice
- El nombre de la relación para la que se crea el índice
- Los atributos sobre los que se define el índice
- El tipo de índice formado

Toda esta información constituye, en efecto, una base de datos en miniatura. Algunos sistemas de bases de datos guardan esta información utilizando estructuras de datos y código especiales. Suele resultar preferible guardar los datos sobre la base de datos en la misma base de datos. Al utilizar la base de datos para guardar los datos del sistema se simplifica la estructura global del sistema y se permite que se utilice toda la potencia de la base de datos en obtener un acceso rápido a los datos del sistema.

La elección exacta de la manera de representar los datos del sistema utilizando relaciones debe tomarla el diseñador del sistema. A continuación se ofrece una representación posible con las claves primarias subrayadas:

Metadatos-catálogo-sistema = (<u>*nombre-relación*</u>, *número-atributos*)
Metadatos-atributos = (<u>*nombre-atributo*</u>, <u>*nombre-relación*</u>, *tipo-dominio*, *posición*, *longitud*)
Metadatos-usuarios = (<u>*nombre-usuario*</u>, *contraseña-cifrada*, *grupo*)
Metadatos-índices = (<u>*nombre-índice*</u>, <u>*nombre-relación*</u>, *tipo-índice*, *atributos-índice*)
Metadatos-vistas = (<u>*nombre-vista*</u>, *definición*)

11.9. ALMACENAMIENTO PARA LAS BASES DE DATOS ORIENTADAS A OBJETOS**

Las técnicas de organización de los archivos descritas en el Apartado 11.7 (como las organizaciones en montículo, secuencial, asociativa y de agrupaciones) también pueden utilizarse para guardar los objetos de las bases de datos orientadas a objetos. Sin embargo, se necesitan características adicionales para poder trabajar con las propiedades de las bases de datos orientadas a objetos, como los campos de conjuntos y los punteros persistentes.

11.9.1. Correspondencia de los objetos con los archivos

La correspondencia de los objetos con los archivos tiene gran parecido con la correspondencia de las tuplas con los archivos de los sistemas relacionales. En el nivel inferior de la representación de los datos, tanto las partes de tuplas de los objetos como las de datos, son sencillamente secuencias de bytes. Por tanto, se pueden guardar los datos de los objetos utilizando las estructuras de archivos descritas en los apartados anteriores con algunas modificaciones que se indican a continuación.

Los objetos de las bases de datos orientadas a objetos pueden carecer de la uniformidad de las tuplas de las bases de datos relacionales. Por ejemplo, los campos de los registros pueden ser conjuntos, a diferencia de las bases de datos relacionales, en los que se suele exigir que los datos se encuentren (por lo menos) en la primera forma normal. Además, los objetos pueden ser muy grandes. Hay que tratar estos objetos de manera diferente de los registros de los sistemas relacionales.

Se pueden implementar campos de conjuntos que tengan un número pequeño de elementos que utilicen estructuras de datos como las listas enlazadas. Los campos de conjuntos que tienen un número de elementos mayor pueden implementarse como relaciones en la base de datos. Los campos de conjuntos también pueden borrarse en el nivel de almacenamiento mediante la normalización: se crea una relación que contenga una tupla para cada valor del campo de conjunto de un objeto. Cada tupla también contiene el identificador del objeto. El sistema de almacenamiento da a los niveles superiores del sistema de bases de datos el aspecto de un campo de conjuntos, aunque en realidad el campo de conjuntos se haya normalizado para crear una relación nueva.

Algunas aplicaciones incluyen objetos muy grandes que no se descomponen fácilmente en componentes menores. Cada uno de estos objetos de gran tamaño puede guardarse en un archivo diferente. Esta idea se discute en el Apartado 11.9.6.

11.9.2. Implementación de los identificadores de los objetos

Dado que los objetos se identifican mediante los identificadores de los objetos (IDO), los sistemas de almacenamiento de objetos necesitan un mecanismo para encontrar un objeto dado su IDO. Si los IDOs son **lógicos** (es decir, no especifican la ubicación del objeto) el sistema de almacenamiento debe tener un índice que asocie los IDOs con la ubicación real del objeto. Si los IDOs son **físicos** (es decir, codifican la ubicación del objeto) se puede encontrar el objeto directamente. Los IDOs físicos suelen tener las tres partes siguientes:

1. Un identificador de volumen o de archivo
2. Un identificador de las páginas dentro del volumen o archivo
3. Un desplazamiento dentro de la página

Además, los IDOs físicos pueden contener un **identificador único**, que es un entero que distingue el IDO de los identificadores de los demás objetos que se hayan guardado anteriormente en la misma ubicación y se borraron o se trasladaron a otra parte. El identificador único también se guarda con el objeto y deben coincidir los identificadores del IDO y los del objeto correspondiente. Si el identificador único de un IDO físico no coincide con el identificador único del objeto al que apunta el IDO, el sistema detecta que el puntero es un puntero colgante e indica un error (un **puntero colgante** es un puntero que no apunta a un objeto válido). En la Figura 11.21 se ilustra este esquema.

Estos errores de puntero tienen lugar cuando se utilizan de manera accidental IDOs físicos correspondientes a objetos antiguos que han sido borrados. Si el espacio ocupado por el objeto se ha vuelto a asignar, puede que haya un objeto nuevo en esa ubicación, y se puede dirigir a él de manera incorrecta el identificador del objeto antiguo. Si no se detecta el uso de los punteros colgantes se puede dar lugar al deterioro del objeto nuevo guardado en la misma ubicación. El identificador único ayuda a detectar estos errores, dado que los

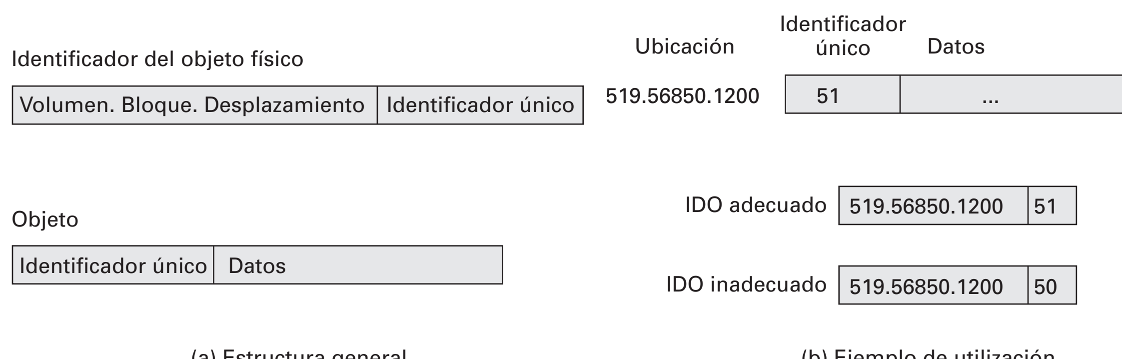


FIGURA 11.21. Identificadores únicos de un IDO.

identificadores únicos del IDO físico antiguo y el del nuevo objeto no coinciden.

Supóngase que hay que desplazar un objeto a una página nueva, quizás debido a que ha aumentado el tamaño del mismo y la página antigua no dispone de espacio adicional. En ese caso, el IDO físico apuntará a la página antigua, que ya no contiene el objeto. En vez de cambiar el IDO del objeto (lo que implica cambiar todos los objetos que apunten hacia él) se deja una **dirección de entrega** en la ubicación antigua. Cuando la base de datos intente encontrar el objeto encontrará la dirección de entrega en su lugar; entonces, utilizará la dirección de entrega para encontrar el objeto.

11.9.3. Gestión de los punteros persistentes

Los punteros persistentes se implementan en un lenguaje de programación persistente utilizando los IDOs. En algunas implementaciones los punteros persistentes son IDOs físicos; en otras, son IDOs lógicos. Una diferencia importante entre los punteros persistentes y los punteros internos de memoria es el tamaño de los mismos. Los punteros internos de memoria sólo necesitan tener el tamaño suficiente para apuntar a toda la memoria virtual. En las computadoras actuales los punteros internos de memoria tienen una longitud de cuatro bytes, que es suficiente para apuntar a cuatro gigabytes de memoria. Las nuevas arquitecturas tienen punteros de ocho bytes.

Los punteros persistentes tienen que apuntar a todos los datos de la base de datos. Dado que los sistemas de bases de datos suelen ser mayores que cuatro gigabytes, los punteros persistentes suelen tener una longitud de al menos ocho bytes. Muchas bases de datos orientadas a objetos proporcionan también identificadores únicos en los punteros persistentes para detectar las referencias colgantes. Esta característica aumenta aún más el tamaño de los punteros persistentes. Por tanto, los punteros persistentes son de una longitud considerablemente mayor que los punteros internos de memoria.

La acción de buscar un objeto dado su identificador se denomina **desreferenciar**. Dado un puntero interno de memoria (como en C++) buscar el objeto es simplemente una referencia a la memoria. Dado un puntero persistente, desreferenciar un objeto tiene una etapa adicional: hay que encontrar la ubicación real del objeto en la memoria buscando el puntero persistente en una tabla. Si el objeto no se halla todavía en la memoria hay que cargarlo desde el disco. Se puede implementar la búsqueda en la tabla de manera bastante eficiente utilizando funciones de asociación, pero la búsqueda sigue siendo lenta comparada con la desreferencia de los punteros, aunque el objeto ya se encuentre en memoria.

El **rescate de punteros** es una manera de reducir el coste de encontrar los objetos persistentes que ya se hallen en la memoria. La idea es que, cuando se desreferencia por primera vez un puntero persistente, se encuentra el objeto y se lleva a la memoria si es que no está ya allí. Ahora se da un paso adicional (se guarda un puntero interno de memoria para el objeto en lugar del puntero persistente). La siguiente ocasión en que se desreferencie el *mismo* puntero persistente la ubicación interna en la memoria puede leerse directamente, por lo que se evita el coste de encontrar el objeto. (En caso de que se puedan volver a desplazar los objetos persistentes de la memoria al disco para hacer sitio para otros objetos persistentes hay que dar otro paso más para asegurarse de que el objeto siga estando en la memoria). De manera análoga, cuando se copia al disco un objeto, hay que **devolver** todos los punteros persistentes que contenga y que se hubieran rescatado y hay que volver a convertirlos en su representación persistente. El rescate de los punteros al efectuar su desreferencia, tal y como se ha descrito aquí, se denomina **rescate software**.

La gestión de la memoria intermedia es más compleja si se utiliza el rescate de punteros, dado que la ubicación física de un objeto no debe cambiar una vez que se lleva ese objeto a la memoria intermedia. Una manera de asegurarse de que no cambiará es clavar las páginas que contengan objetos rescatados en la memoria intermedia, de manera que no sean sustituidas hasta que el programa que realizó el rescate haya concluido. Véanse las notas bibliográficas para tener información sobre esquemas de gestión de la memoria intermedia más complejos, que incluyen técnicas de correspondencia de la memoria virtual, que eliminan la necesidad de clavar las páginas de la memoria intermedia.

11.9.4. Rescate hardware

La existencia de dos tipos de punteros, persistentes y transitorios (internos de la memoria), resulta poco conveniente. Los programadores tienen que recordar el tipo de puntero y puede que tengan que escribir el código dos veces (una para los punteros persistentes y otra para los punteros internos de la memoria). Resultaría más conveniente que tanto los punteros persistentes como los internos de la memoria fueran del mismo tipo.

Una manera sencilla de combinar los tipos de puntero persistente e interno de la memoria es extender simplemente la longitud de los punteros internos de la memoria hasta el mismo tamaño que tienen los punteros persistentes y utilizar un bit del identificador para distinguir entre punteros persistentes e internos de la memoria. Sin embargo, el coste de almacenamiento de los punteros persistentes de mayor longitud tienen que soportarlo también los punteros internos de la memoria; por tanto, este esquema no se utiliza mucho.

Se describirá una técnica denominada **rescate hardware** que utiliza el hardware para la gestión de la memoria presente en la mayor parte de los sistemas informáticos actuales para abordar este problema. Cuando se accede a los datos de una página en memoria virtual y el sistema operativo detecta que la página no tiene almacenamiento real asignado, o que ha sido protegida contra accesos, entonces ocurre una **violación de la segmentación**[3]. Muchos sistemas operativos proporcionan

un mecanismo para especificar una función a la que se llama cuando sucede una violación de segmentación, y también un mecanismo para asignar almacenamiento para una página en el espacio virtual de direcciones y para establecer los permisos de acceso a la página. En la mayoría de sistemas Unix, la llamada al sistema mmap proporciona esta última característica. El rescate hardware hace un uso inteligente de los mecanismos anteriores.

El rescate hardware presenta dos ventajas fundamentales respecto al rescate software.

1. Puede guardar los punteros persistentes en los objetos en el mismo espacio que necesitan los punteros internos de la memoria (junto con el almacenamiento adicional externo al objeto).
2. Transforma de manera transparente los punteros persistentes en internos de la memoria, y viceversa, de forma inteligente y eficiente. El software escrito para trabajar con los punteros internos de la memoria puede, por tanto, trabajar también con los punteros persistentes sin que haya necesidad de efectuar ningún cambio.

11.9.4.1. Representación de punteros

El rescate hardware utiliza la siguiente representación de los punteros persistentes contenidos en los objetos que se hallan en el disco. Un puntero persistente se divide conceptualmente en dos partes: un identificador de página en la base de datos y un desplazamiento en la misma página 4. El identificador de la página es en realidad un puntero indirecto de pequeño tamaño: cada página (u otra unidad de almacenamiento) tiene una tabla de traducción que proporciona una asociación entre los identificadores de página cortos y los identificadores de página completos de la base de datos. El sistema tiene que buscar el identificador de página corto en los punteros persistentes de la tabla de traducción para encontrar el identificador de página completo.

La tabla de traducción, en el peor de los casos, sólo tendrá un tamaño igual que el número máximo de punteros que puedan contener los objetos de una página; con un tamaño de página de 4.096 y un tamaño de puntero de cuatro bytes el número máximo de punteros es 1.024. En la práctica, la tabla de traducción probablemente contenga muchos menos elementos que el número máximo (1.024 en este ejemplo) y no ocupe demasiado espacio. El identificador de página corto sólo tiene que tener los bytes necesarios para identificar una fila de la tabla; con un tamaño de tabla máximo de 1.024 sólo hacen falta diez bytes. Por tanto, basta con un número pequeño de bits para guardar el identificador de página corto. Por tanto, la tabla de traducción permite que los punteros persistentes que no sean cortos quepan en el mismo espacio que los punteros internos de la memoria. Aunque sólo hacen falta unos pocos bits para los identificadores de página cortos, se utilizan como tales todos los bits de los punteros internos de la memoria distintos de los de desplazamiento de página. Esta arquitectura, como se verá, facilita el rescate.

La representación del esquema de punteros persistentes se muestra en la Figura 11.22, en la que hay tres objetos en la página, cada uno de los cuales contiene un puntero persistente. La tabla de traducción muestra la asociación entre los identificadores de página cortos y los identificadores de página sin abreviar de la base de datos para cada uno de los identificadores de página cortos de estos punteros persistentes. Los identificadores de página de la base de datos se muestran en el formato *volumen.página.desplazamiento*.

Se guarda información adicional con cada página para que se puedan encontrar todos los punteros persistentes de la página. El sistema actualiza la información cuando se crea o borra algún objeto de la página. La necesidad de encontrar todos los punteros persistentes de cada página se pondrá de manifiesto más adelante.

11.9.4.2. Rescate de punteros en una página

Inicialmente no se ha iniciado ninguna página en memoria virtual. Las páginas de la memoria virtual se pueden asignar a las páginas de la base de datos antes de que realmente se carguen, como se verá enseguida. Las páginas de la base de datos se cargan en memoria virtual cuando el sistema de bases de datos necesite acceder a los datos de la página. Antes de que se cargue una página de la base de datos, el sistema asigna una página de memoria virtual para ella si no se hubiese asignado ya una. El sistema carga la página de la base de datos en la página de la memoria virtual que se ha asignado.

Cuando el sistema carga una página P de la base de datos en memoria virtual, se realiza el rescate de punteros en la página: se buscan todos los punteros persistentes contenidos en los objetos de la página P utilizando la información adicional guardada en la misma. Para cada puntero en la página se realizan las siguientes acciones (sea $\langle p_i, d_i \rangle$ el valor del puntero persistente, donde p_i es el identificador de página corto y d_i es el desplazamiento de la página, y sea P_i el identificador de página completo de p_i, hallado en la tabla de traducción de la página P).

[3] A veces se usa el término **fallo de página** en lugar de *violación de la segmentación*, aunque las violaciones de protección de acceso no se consideran como fallos de página.

[4] El término página se usa normalmente para referirse a la página de memoria real o virtual, y el término bloque se usa para referirse a los bloques de disco de la base de datos. En el rescate hardware deben ser del mismo tamaño y los bloques de la base de datos se buscan en las páginas de la memoria virtual. Se usarán los términos página y bloque para significar lo mismo.

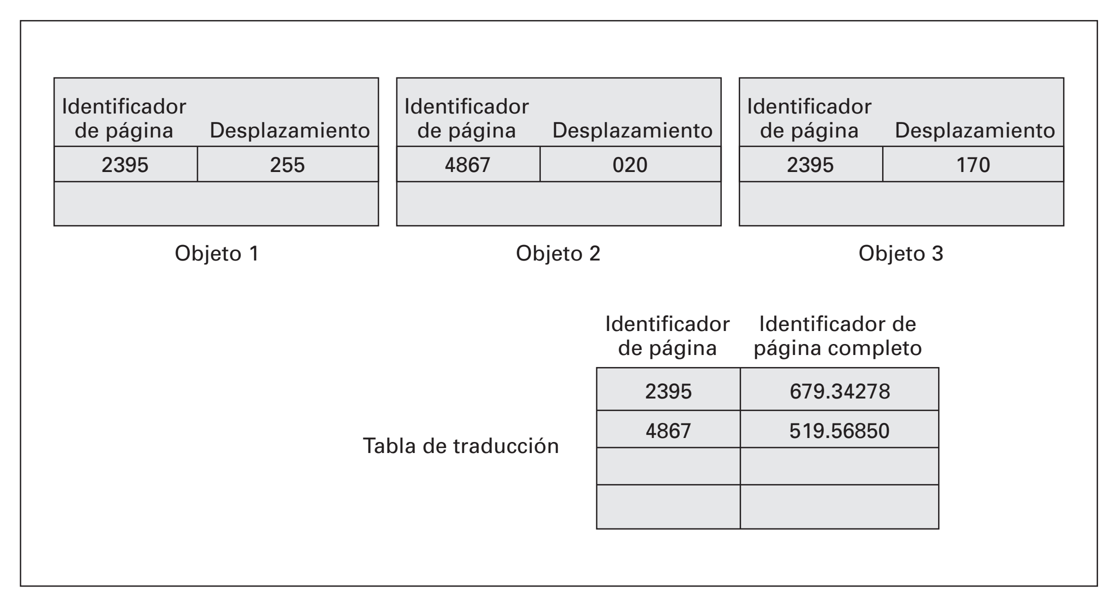


FIGURA 11.22. Imagen de una página antes del rescate.

1. Si la página P_i no tiene asignada todavía una página de memoria virtual se le asignará ahora una página libre del espacio de las direcciones virtuales. La página P_i residirá en la ubicación de la dirección virtual en el momento en que se lleve si esto se lleva a cabo. En este momento la página del espacio de direcciones virtuales no tiene espacio de almacenamiento asignado, ni en la memoria ni en el disco; se trata simplemente de un rango de direcciones reservado para la página de la base de datos. El sistema asigna espacio real cuando realmente carga la página P_i de la base de datos en memoria virtual.
2. Sea v_i la página de la memoria virtual asignada a P_i (bien con anterioridad, bien en el paso anterior). El sistema actualiza el puntero persistente considerado, cuyo valor es $\langle p_i, d_i \rangle$, reemplazando p_i por v_i.

Después de actualizar todos los punteros persistentes, cada entrada $\langle p_i, P_i \rangle$ de la tabla de traducción se reemplaza por $\langle v_i, P_i \rangle$, donde v_i es la página de memoria virtual que se ha asignado para P_i.

En la Figura 11.23 se muestra el estado de la página de la Figura 11.22 después de que se haya llevado a la memoria y se hayan rescatado sus punteros. Aquí se supone que la página cuyo identificador de página de la base de datos es 679.34278 se ha asociado con la página 5001 de la memoria, mientras que la página cuyo identificador es 519.56850 se ha hecho corresponder con la página 4867 (que coincide con el identificador de página corto). Todos los punteros de los objetos se han actua-

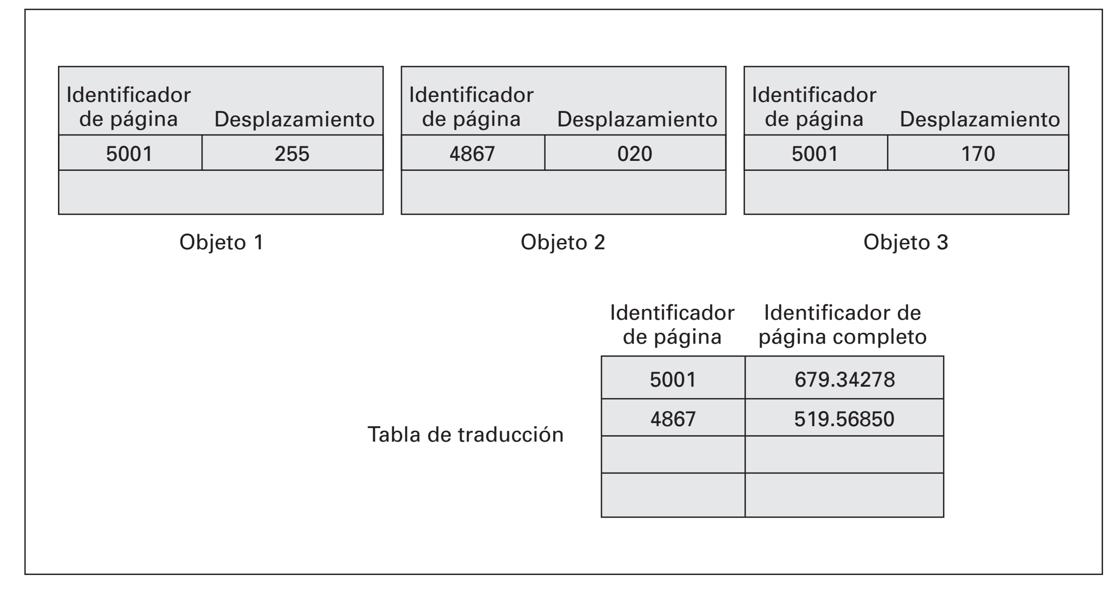


FIGURA 11.23. Imagen de la página después del rescate.

lizado para que reflejen la nueva asociación y pueden utilizarse como punteros internos de la memoria.

Al final de la fase de traducción de cada página, los objetos de la misma cumplen una propiedad importante: todos los punteros persistentes contenidos en los objetos de esa página se han transformado en punteros internos de la memoria. Por tanto, los objetos de las páginas internas de la memoria sólo contienen punteros internos de la memoria. Las rutinas que utilicen estos objetos ni siquiera tienen que conocer la existencia de los punteros persistentes. Por ejemplo, las bibliotecas existentes escritas para los objetos internos de la memoria pueden utilizarse sin modificación alguna para los objetos persistentes. Esto es una ventaja importante.

11.9.4.3. Desreferencia de punteros

Considérese la primera vez que se desreferencia un puntero interno de la memoria de una página v, cuando todavía no se ha asignado espacio de almacenamiento para esa página. Como se ha descrito, tendrá lugar una violación del encauzamiento y dará lugar a una llamada a una función en el sistema de bases de datos. El sistema de bases de datos realiza las siguientes acciones:

1. En primer lugar determina la página de la base de datos que se asignó a la página de la memoria virtual v_i; sea P_i el identificador de página sin acortar de la página de la base de datos (si no hay ninguna página de la base de datos asignada a v_i se indicará la existencia de un error).
2. Asigna espacio de almacenamiento para la página v_i y se carga en ella la página P_i de la base de datos.
3. Realiza rescate de punteros en la página P_i, el sistema permite que continúe la desreferencia del puntero que resultó en la violación de segmentación. La desreferencia del puntero encontrará cargado en memoria el objeto que estaba buscando.

Si cualquier puntero rescatado que apunte a un objeto en la página v_i se desreferencia más tarde, la desreferencia funciona igual que cualquier otro acceso a la memoria virtual, sin sobrecargas extra. En cambio, si no se usa el rescate, hay una considerable sobrecarga al localizar la página de la memoria intermedia que contiene el objeto y su posterior acceso. Esta sobrecarga aparece en *cada* acceso a los objetos de la página mientras que, cuando se usa el rescate, la sobrecarga sólo aparece en el *primer* acceso al objeto en la página. Los accesos posteriores funcionan a la velocidad normal de los accesos a memoria virtual. Por tanto, el rescate hardware proporciona excelentes beneficios de rendimiento para aplicaciones que desreferencien punteros repetidamente.

11.9.4.4. Optimizaciones

El rescate software tiene una operación de devolución asociada, cuando hay que volver a escribir en la base de datos una página de la memoria, para transformar de nuevo los punteros internos de la memoria en persistentes. El rescate hardware puede incluso evitar este paso (cuando se realiza el rescate de punteros de la página sencillamente se actualiza la tabla de traducción, por lo que la parte del identificador de página de los punteros internos de la memoria simulados puede utilizarse para buscar en la tabla). Por ejemplo, tal y como se muestra en la Figura 11.23, la página 679.34278 de la base de datos (con el identificador corto 2395 en la página mostrada) se asocia con la página 5001 de la memoria virtual. En este momento no sólo se actualiza el puntero del objeto 1 de 2395255 a 5001255, sino que también se actualiza a 5001 el identificador corto de la tabla. Por tanto, el identificador corto 5001 del objeto 1 y de la tabla vuelven a coincidir. Por consiguiente, se puede volver a escribir la página en el disco sin necesidad de devolverla.

Se pueden llevar a cabo varias optimizaciones del esquema básico aquí mostrado. Cuando se realiza el rescate de la página P, el sistema intenta asignar P' a la posición de dirección virtual indicada por el identificador de página corto de P' de la página P. Si la página puede asignarse tal y como se ha intentado, no hay que actualizar los punteros que la apunten. En el ejemplo de rescate expuesto, la página 519.56850 con el identificador de página corto 4867 se asoció con la página 4867 de la memoria virtual, que coincide con su identificador de página corto. Puede verse que no hay que modificar el puntero del objeto 2 que apunta a esta página durante el rescate. Si puede asignarse cada página a su posición correcta en el espacio de las direcciones virtuales no habrá que transformar ninguno de los punteros y se reducirá de modo significativo el coste del rescate.

El rescate hardware funciona aunque la base de datos sea de mayor tamaño que la memoria virtual, pero sólo mientras todas las páginas a las que cada proceso concreto tenga acceso quepan en la memoria virtual del mismo. Si no caben, habrá que sustituir las páginas llevadas a la memoria virtual, y esa sustitución resulta difícil, dado que puede haber punteros internos de la memoria que apunten a objetos de esas páginas. También puede utilizarse teóricamente el rescate hardware en el nivel de los conjuntos de páginas (a menudo denominados segmentos), en lugar de para páginas aisladas, siempre que los identificadores de página cortos, con los desplazamientos de página, no superen la memoria de los punteros internos de la memoria.

El rescate hardware también se puede usar en el nivel de los conjuntos de páginas (a menudo denominados segmentos) en lugar de en una sola página. Para el rescate por conjuntos el sistema usa una única tabla de traducción para todas las páginas del segmento. Carga las páginas en el segmento y las rescata cuando es necesario; no es necesario que se carguen todas juntas.

11.9.5. Estructura de los objetos en el disco o en la memoria

El formato con el que se guardan los objetos en la memoria puede ser diferente del formato con el que se guardan en el disco en la base de datos. Un motivo puede ser el uso del rescate software, en el que la estructura de los punteros persistentes y la de los internos de la memoria son diferentes. Otro motivo puede ser que se desee hacer que la base de datos sea accesible desde diferentes máquinas, posiblemente basadas en arquitecturas diferentes, y desde lenguajes diferentes, y desde programas compilados con compiladores diferentes, todo lo cual da lugar a diferencias en la representación en la memoria.

Considérese, por ejemplo, la definición de una estructura de datos en un lenguaje de programación como C++. La estructura física (como los tamaños y la representación de los enteros) del objeto es dependiente de la máquina en la que se ejecuta el programa[5]. Además, la estructura física puede depender también del compilador que se utilice; en un lenguaje tan complejo como C++ son posibles diferentes opciones para la traducción de la descripción de nivel superior a la estructura física, y cada compilador puede tomar sus propias opciones.

La solución a este problema es hacer independiente de la máquina y del compilador la representación física de los objetos de la base de datos. Los objetos pueden pasarse de la representación en el disco a las formas necesarias para la máquina, lenguaje y compilador concretos cuando se llevan a la memoria. Esta conversión puede hacerse de manera transparente al mismo tiempo que se rescatan los punteros del objeto, de modo que el programador no tenga que preocuparse por ella.

El primer paso en la implementación de un esquema así es definir un lenguaje común para describir la estructura de los objetos (es decir, un lenguaje para la definición de datos). Se han realizado varias propuestas, una de las cuales es el lenguaje de definición de objetos (Object Definition Language, ODL) desarrollado por el grupo de gestión de bases de datos de objetos (Object Database Management Group, ODMG). ODL tiene definidas asociaciones con Java, C++ y Smalltalk, por lo que en teoría los objetos de una base de datos que cumpla ODMG se pueden tratar utilizando cualquiera de estos lenguajes.

La definición de la estructura de cada clase de la base de datos se guarda (de manera lógica) en las bases de datos. El código para pasar los objetos de la base de datos a la representación de los mismos que trata el lenguaje de programación (y viceversa) es dependiente de la máquina y del compilador del lenguaje. Este código se puede generar de manera automática utilizando las definiciones de las clases de los objetos guardadas previamente.

Una fuente inesperada de diferencias entre las representaciones de los datos en el disco y en la memoria son los punteros ocultos de los objetos. Los **punteros ocultos** son punteros transitorios que generan los compiladores y se guardan en los objetos. Estos punteros apuntan (de modo indirecto) a las tablas utilizadas para implementar ciertos métodos del objeto. Las tablas suelen compilarse en código objeto ejecutable y la ubicación exacta de las tablas depende del código objeto ejecutable, por lo que puede ser diferente en procesos diferentes. Por tanto, cuando un proceso tiene acceso a un objeto, los punteros ocultos deben fijarse para que apunten a la ubicación correcta. Los punteros ocultos pueden inicializarse al tiempo que se llevan a cabo las conversiones entre las representaciones de los datos.

11.9.6. Objetos de gran tamaño

Los objetos pueden ser también enormemente grandes; por ejemplo, los objetos multimedia pueden ocupar varios megabytes. Los elementos de datos excepcionalmente grandes, como las secuencias de vídeo, pueden llegar a los gigabytes, aunque suelen dividirse en varios objetos, cada uno de ellos del orden de unos pocos megabytes o menos. Los objetos de gran tamaño que contienen datos binarios se denominan objetos de gran tamaño en binario (binary large objects, blobs), mientras que los grandes objetos que contienen datos de caracteres se denominan objetos de gran tamaño de tipo carácter (character large objects, clobs), como se vio en el Apartado 9.2.1.

La mayor parte de las bases de datos relacionales limitan el tamaño de los registros para que no tengan una longitud mayor que el tamaño de la página para simplificar la gestión de la memoria intermedia y del espacio libre. Los objetos de gran tamaño y los campos largos suelen guardarse en un archivo especial (o en un conjunto de archivos) reservado para el almacenamiento de campos largos.

Se presenta un problema en la gestión de los objetos de gran tamaño con la asignación de las páginas de

[5] Por ejemplo, las arquitecturas Motorola 680×0, la arquitectura IBM 360 y las arquitecturas Intel 80386/80486/Pentium/Pentium-II/Pentium-III tienen todas enteros de cuatro bytes. Sin embargo, se diferencian en la manera en que se disponen en las palabras los bits de los enteros. En las computadoras personales de las primeras generaciones los enteros tenían una longitud de dos bytes; en las arquitecturas de estaciones de trabajo más recientes, como la Alpha de Compaq, Itanium de Intel y UltraSparc de Sun, los enteros pueden tener una longitud de ocho bytes.

memoria intermedia. Los objetos de gran tamaño pueden necesitar ser guardados en una secuencia contigua de bytes cuando se llevan a la memoria; en este caso, si un objeto es mayor que una página, se deben asignar páginas contiguas de la memoria intermedia para guardarlo, lo que hace más difícil la gestión de la memoria intermedia.

Los objetos de gran tamaño suelen modificarse actualizando una parte de los mismos o insertándoles o borrándoles alguna parte del objeto, en lugar de escribiendo todo el objeto. Si hay que trabajar con inserciones o borrados, se pueden implementar los objetos de gran tamaño utilizando estructuras de árboles B (que se estudian en el Capítulo 12). Las estructuras de árboles B permiten leer todo el objeto, así como insertar y borrar partes del mismo.

Por razones prácticas se pueden manipular los objetos de gran tamaño utilizando programas de aplicaciones en vez de hacerlo dentro de la base de datos:

- **Datos de texto.** El texto suele tratarse como una cadena de bytes con la que trabajan los editores y los formateadores.
- **Datos gráficos.** Los datos gráficos pueden representarse como mapas de bits o como conjuntos de líneas, cuadros y otros objetos geométricos. Aunque algunos datos gráficos suelen tratarse dentro de la base de datos, en muchos casos se utiliza software de aplicaciones especiales, como en el diseño de circuitos integrados.
- **Datos de sonido y de vídeo.** Los datos de sonido y de vídeo suelen ser una representación digitalizada y comprimida creada y reproducida por software de aplicaciones diferentes. La modificación de los datos suele realizarse con software especial para ediciones, fuera del sistema de bases de datos.

El método más utilizado para actualizar estos datos es el método **desmarcar/marcar**. El usuario o la aplicación **desmarca** una copia de un objeto de campo largo, opera con esta copia utilizando programas especiales de aplicaciones y luego **marca** la copia modificada. Los conceptos *desmarcar* y *marcar* se corresponden a grandes rasgos con los de lectura y escritura. En algunos sistemas, al marcar se puede crear una nueva versión del objeto sin borrar la anterior.

11.10. RESUMEN

- En la mayor parte de los sistemas informáticos hay varios tipos de almacenamiento de datos. Estos medios de almacenamiento se clasifican según la velocidad con la que se puede tener acceso a los datos, el coste de adquisición de la memoria por unidad de datos y su fiabilidad. Entre los medios disponibles suelen estar la organización caché, la memoria principal, la memoria *flash*, los discos magnéticos, los discos ópticos y las cintas magnéticas.
- La fiabilidad de los medios de almacenamiento se determina mediante dos factores: si un corte en el suministro eléctrico o una caída del sistema hace que los datos se pierdan, y la probabilidad de fallo físico del dispositivo de almacenamiento.
- Se puede reducir la probabilidad del fallo físico conservando varias copias de los datos. Para los discos se puede utilizar la creación de imágenes. También se pueden usar métodos más sofisticados como las disposiciones redundantes de discos independientes (RAID). La distribución de los datos entre los discos ofrece altos índices de productividad en los accesos de gran tamaño; introduciendo la redundancia entre los discos se mejora mucho la fiabilidad. Se han propuesto varias organizaciones RAID diferentes, con características de coste, rendimiento y fiabilidad diferentes. Las organizaciones RAID de nivel 1 (la creación de imágenes) y RAID nivel 5 son las más utilizadas.
- Los *archivos* se pueden organizar lógicamente como una secuencia de registros asociados con bloques de disco. Un enfoque de la asociación de la base de datos con los archivos es utilizar varios archivos y guardar los registros de una única longitud fija en cualquier archivo dado. Una alternativa es estructurar los archivos de modo que puedan aceptar registros de longitud variable. Hay diferentes técnicas para la implementación de los registros de longitud variable, incluyendo el método de la página con ranuras, el método de los punteros y el método del espacio reservado.
- Dado que los datos se transfieren entre el almacenamiento en disco y la memoria principal en unidades de bloques, merece la pena asignar los registros de los archivos de modo que cada bloque contenga registros relacionados. Si se puede tener acceso a varios de los registros deseados utilizando sólo un acceso a bloques se evitan accesos al disco. Dado que los accesos al disco suelen ser el cuello de botella del rendimiento de los sistemas de bases de datos, la esmerada asignación de los registros a los bloques puede ofrecer mejoras significativas del rendimiento.
- Una manera de reducir el número de accesos al disco es guardar todos los bloques posibles en la memoria principal. Dado que no se pueden guardar todos los bloques en la memoria principal, hay que gestionar la asignación del espacio disponible en la memoria principal para el almacenamiento de los bloques. La

memoria intermedia (*buffer*) es la parte de la memoria disponible para el almacenamiento de las copias de los bloques del disco. El subsistema responsable de la asignación del espacio de la memoria intermedia se denomina *gestor de la memoria intermedia*.

- Los sistemas de almacenamiento para las bases de datos orientadas a objetos son algo diferentes de los sistemas de almacenamiento para las bases de datos relacionales: deben trabajar con objetos de gran tamaño, por ejemplo, y con punteros persistentes. Hay esquemas para detectar los punteros persistentes colgantes.
- Los esquemas de rescate software y hardware permiten la desreferencia eficiente de los punteros persistentes. Los esquemas basados en hardware utilizan el apoyo a la gestión de la memoria virtual realizado en hardware y muchos sistemas operativos de la generación actual los hacen accesibles a los programas del usuario.

TÉRMINOS DE REPASO

- Almacenamiento terciario
  — Discos ópticos
  — Cintas magnéticas
  — Cambiadores automáticos
- Archivo
- Bloque de disco
- Catálogo del sistema
- Clave de búsqueda
- Diccionario de datos
- Disco magnético
  — Plato
  — Discos rígidos
  — Disquetes
  — Pistas
  — Sectores
  — Cabeza de lectura y escritura
  — Brazo del disco
  — Cilindro
  — Controlador de discos
  — Comprobación de suma
  — Reasignación de sectores defectuosos
- Disposición redundante de discos independientes (RAID)
  — Creación de imágenes
  — Distribución de datos
  — Distribución en el nivel de bit
  — Distribución en el nivel de bloque
- Estructuras de almacenamiento para BDOO
- Identificador del objeto (IDO)
  — IDO lógico
  — IDO físico
  — Identificador único
  — Puntero colgante
  — Dirección de entrega
- Intercambio en caliente
- Medidas de rendimiento de los discos
  — Tiempo de acceso
  — Tiempo de búsqueda
  — Latencia rotacional
  — Velocidad de transferencia de datos
  — Tiempo medio entre fallos
- Medios de almacenamiento físico
  — Caché
  — Memoria principal
  — Memoria flash
  — Disco magnético
  — Almacenamiento óptico
- Memoria intermedia (buffer)
  — Gestor de la memoria intermedia
  — Bloques clavados
  — Salida forzada de bloques
- Niveles de RAID
  — Nivel 0 (distribución de bloques sin redundancia)
  — Nivel 1 (distribución de bloques con creación de imágenes)
  — Nivel 3 (distribución de bits con paridad)
  — Nivel 5 (distribución de bloques con paridad distribuida)
  — Nivel 6 (distribución de bloques con redundancia P + Q)
- Objetos de gran tamaño
- Optimización del acceso a bloques de disco
  — Planificación del brazo
  — Algoritmo del ascensor
  — Organización de archivos
  — Desfragmentación
  — Memorias intermedias de escritura no volátiles
  — Memoria no volátil de acceso aleatorio
  — Disco del registro histórico
  — Sistema de archivos basado en registro histórico

- Organización asociativa (hash) de archivos
- Organización de archivos
  - Lista libre
- Organización de archivos en agrupaciones
- Organización de archivos en montículo
- Organización de archivos secuenciales
- Políticas de sustitución de la memoria intermedia
  - Menos recientemente utilizado (Least Recently Used, LRU)
  - Extracción inmediata
  - Más recientemente utilizado (Most Recently Used, MRU)
- Punteros ocultos
- RAID hardware
- RAID software
- Registros de longitud fija
  - Representación en cadenas de bytes
  - Estructura de páginas con ranuras
  - Espacio reservado
  - Representación de listas
- Registros de longitud variable
  - Cabecera del archivo
  - Lista libre
- Rendimiento de la reconstrucción
- Rescate de punteros
  - Desreferencia
  - Devolución
  - Rescate software
  - Rescate hardware
  - Violación de la segmentación
  - Fallo de página

EJERCICIOS

11.1. Indíquense los medios de almacenamiento físico disponibles en las computadoras que se utilizan habitualmente. Dese la velocidad con la que se puede tener acceso a los datos en cada medio.

11.2. ¿Cómo afecta la reasignación de los sectores dañados por los controladores de disco a la velocidad de recuperación de los datos?

11.3. Considérese la siguiente disposición de los bloques de datos y de paridad de cuatro discos:

Disco 1	Disco 2	Disco 3	Disco 4
B_1	B_2	B_3	B_4
P_1	B_5	B_6	B_7
B_8	P_2	B_9	B_{10}
⋮	⋮	⋮	⋮

B_i representa los bloques de datos; P_i, los bloques de paridad. El bloque de paridad P_i es el bloque de paridad para los bloques de datos B_{4i-3} a B_{4i}. Indíquense los problemas que puede presentar esta disposición.

11.4. Un fallo en el suministro eléctrico que se produzca mientras se escribe un bloque del disco puede dar lugar a que el bloque sólo se escriba parcialmente. Supóngase que se pueden detectar los bloques escritos parcialmente. Un proceso atómico de escritura de bloque es aquel en el que se escribe el bloque entero o no se escribe nada (es decir, no hay procesos de escritura parciales). Propónganse esquemas para conseguir el efecto de los procesos atómicos de escritura con los siguientes esquemas RAID. Los esquemas deben implicar procesos de recuperación de fallos.

a. RAID de nivel 1 (creación de imágenes)
b. RAID de nivel 5 (entrelazado de bloques, paridad distribuida)

11.5. Los sistemas RAID suelen permitir la sustitución de los discos averiados sin que se impida el acceso al sistema. Por tanto, los datos del disco averiado deben reconstruirse y escribirse en el disco de repuesto mientras el sistema se halla en funcionamiento. ¿Con cuál de los niveles RAID es menor la interferencia entre los accesos al disco reconstruido y los accesos al resto de los discos? Justifíquese la respuesta.

11.6. Dese un ejemplo de una expresión de álgebra relacional y de una estrategia de procesamiento de consultas en cada una de las situaciones siguientes:

a. MRU es preferible a LRU.
b. LRU es preferible a MRU.

11.7. Considérese el borrado del registro 5 del archivo de la Figura 11.8. Compárense las ventajas relativas de las siguientes técnicas para implementar el borrado:

a. Trasladar el registro 6 al espacio ocupado por el registro 5 y desplazar el registro 7 al espacio ocupado por el registro 6.
b. Trasladar el registro 7 al espacio ocupado por el registro 5.
c. Marcar el registro 5 como borrado y no desplazar ningún registro.

11.8. Muéstrese la estructura del archivo de la Figura 11.9 después de cada uno de los pasos siguientes:

a. Insertar (C-323, Galapagar, 1600).
b. Borrar el registro 2.
c. Insertar (C-626, Galapagar, 2000).

11.9. Dese un ejemplo de una aplicación de bases de datos en que sea preferible el método del espacio reservado para la representación de los registros de longitud variable frente al método de los punteros. Justifíquese la respuesta.

11.10. Dese un ejemplo de una aplicación de bases de datos en la que sea preferible el método de los punteros para representar los registros de longitud variable al método del espacio reservado. Justifíquese la respuesta.

11.11. Muéstrese la estructura del archivo de la Figura 11.12 después de cada uno de los pasos siguientes:

a. InsertFar (C-101, Becerril, 2800).

b. Insertar (C-323, Galapagar, 1600).

c. Borrar (C-102, Navacerrada, 400).

11.12. ¿Qué ocurre si se intenta insertar el registro

(C-929, Navacerrada, 3000)

en el archivo de la Figura 11.12?

11.13. Muéstrese la estructura del archivo de la Figura 11.13 después de cada uno de los pasos siguientes:

a. Insertar (C-101, Becerril, 560.000).

b. Insertar (C-323, Galapagar, 320.000).

c. Borrar (C-102, Navacerrada, 80.000).

11.14. Explíquese por qué la asignación de los registros a los bloques afecta de manera significativa al rendimiento de los sistemas de bases de datos.

11.15. Si es posible, determínese la estrategia de gestión de la memoria intermedia de su sistema operativo ejecutándose en su computadora y los mecanismos que proporciona para controlar la sustitución de páginas. Discútase cómo el control sobre la sustitución que proporciona podría ser útil para la implementación de sistemas de bases de datos.

11.16. En la organización secuencial de los archivos, ¿por qué se utiliza un bloque de desbordamiento aunque sólo haya en ese momento un único registro de desbordamiento?

11.17. Indíquense dos ventajas y dos inconvenientes de cada una de las estrategias siguientes para el almacenamiento de bases de datos relacionales:

a. Guardar cada relación en un archivo.

b. Guardar varias relaciones (quizá toda la base de datos) en un archivo.

11.18. Considérese una base de datos relacional con dos relaciones:

curso (*nombre-curso*, *aula*, *profesor*)
matrícula (*nombre-curso*, *nombre-estudiante*, *nivel*)

Defínanse ejemplos de estas relaciones para tres cursos, en cada uno de los cuales se matriculan cinco estudiantes. Dese una estructura de archivos de estas relaciones que utilice la agrupación.

11.19. Considérese la siguiente técnica de mapa de bits para realizar el seguimiento del espacio libre de un archivo. Por cada bloque del archivo se mantienen dos bits en el mapa. Si el bloque está lleno entre el 0 y el 30 por ciento, los bits son 00, entre 30 por ciento y 60 por ciento, 01, entre 60 por ciento y 90 por ciento, 10, y por encima de 90 por ciento, 11. Tales mapas se pueden mantener en memoria incluso para grandes archivos.

a. Descríbase cómo mantener actualizado el mapa de bits al insertar y eliminar registros.

b. Descríbanse el beneficio de la técnica de los mapas de bits sobre las listas libres al buscar espacio libre y al actualizar su información.

11.20. Dese una versión normalizada de la relación *Metadatos-índices* y explíquese por qué al usar la versión normalizada se incurriría en pérdida de rendimiento.

11.21. Explíquese el motivo de que un IDO físico deba contener más información que un puntero que apunte a una ubicación física de almacenamiento.

11.22. Si se utilizan IDOs físicos, un objeto se puede reubicar guardando un puntero a su nueva ubicación. En el caso de que se guarden varios punteros para un objeto, ¿cuál sería el efecto sobre la velocidad de recuperación?

11.23. Defínase el término *puntero colgante*. Descríbase la manera en que el esquema de identificador único ayuda a detectar los punteros colgantes en las bases de datos orientadas a objetos.

11.24. Considérese el ejemplo de la página 276, que muestra que no hace falta el rescate si se utiliza el rescate hardware. Explíquese el motivo de que, en ese ejemplo, resulte seguro cambiar el identificador corto de la página 679.34278 de 2395 a 5001. ¿Puede tener ya alguna otra página el identificador corto 5001? Si fuera así, ¿cómo se resolvería esa situación?

NOTAS BIBLIOGRÁFICAS

Patterson y Hennessy [1995] discuten los aspectos de hardware de la memoria intermedia con traducción anticipada, de las cachés y de las unidades de gestión de la memoria. Rosch y Wethington [1999] presentan una excelente visión general del hardware de computadoras, incluyendo un tratamiento extensivo de todos los tipos de tecnologías de almacenamiento como disquetes, discos magnéticos, discos ópticos, cintas e interfaces de almacenamiento. Ruemmler y Wilkes [1994] presentan una revisión de lo tecnología de los discos magnéticos. La memoria flash se discute en Dippert y Levy [1993].

Las especificaciones de las unidades de disco actuales se pueden obtener de los sitios Web de sus fabricantes, como IBM, Seagate y Maxtor.

Las organizaciones alternativas de los discos que proporcionan un elevado grado de tolerancia a los fallos incluyen las desarrolladas por Gray et al. [1990] y por Bitton y Gray [1988]. La distribución en los discos se describe en Salem y García-Molina [1986]. Se presentan discusiones sobre las disposiciones redundantes de discos independientes (RAID) en Patterson et al. [1988] y en Chen y Patterson [1990]. Chen et al. [1994] pre-

sentan una excelente revisión de los principios y de la aplicación de RAID. Los códigos de Reed-Solomon se tratan en Pless [1989]. El sistema de archivos basado en registro histórico, que hace secuencial el acceso a disco, se describe en Rosenblum y Ousterhout [1991].

En los sistemas que permiten la informática portátil se pueden transmitir los datos de manera reiterada. El medio de transmisión puede considerarse un nivel de la jerarquía de almacenamiento (como un disco transmisor con latencia elevada). Estos aspectos se discuten en Acharya et al. [1995]. La gestión de la caché y de las memorias intermedias en la informática portátil se discute en Barbará e Imielinski [1994]. En Douglis et al. [1994] aparecen más discusiones de los problemas de almacenamiento en la informática portátil.

Las estructuras de datos básicas se discuten en Cormen et al.[1990]. Hay varios artículos que describen la estructura de almacenamiento de sistemas específicos de bases de datos. Astrahan et al. [1976] y System R. Chamberlin et al. [1981] repasan en retrospectiva, System R. *Oracle 8 Concepts Manual* (Oracle [1997]) describe la organización de almacenamiento del sistema de bases de datos Oracle 8. La estructura de Wisconsin Storage System (WiSS) se describe en Chou et al. [1985]. En Finkelstein et al. [1988] se describe una herramienta de software para el diseño físico de bases de datos relacionales.

La gestión de las memorias intermedias se discute en la mayor parte de los textos sobre sistemas operativos, incluido Silberschatz y Galvin [1994]. Stonebraker [1981] discute la relación entre los gestores de memoria intermedia de los sistemas de bases de datos y los de los sistemas operativos. Chou y DeWitt [1985] presentan algoritmos para la gestión de memoria intermedia en sistemas de bases de datos y describe un método de medida del rendimiento. Bridge et al. [1997] describen técnicas usadas en el gestor de la memoria intermedia del sistema de bases de datos Oracle.

En Wilson [1990], Moss [1990] y White y DeWitt [1992] se ofrecen descripciones y comparaciones del rendimiento de diferentes técnicas de rescate. White y DeWitt [1994] describen el esquema de gestión de memoria intermedia asociado con la memoria virtual utilizado en el sistema ObjectStore OODB y en el gestor de almacenamiento QuickStore. Utilizando este esquema se pueden asociar las páginas del disco con direcciones fijas de la memoria virtual, aunque no estén clavadas en la memoria intermedia. El gestor de almacenamiento de objetos Exodus se describe en Carey et al. [1986]. Biliris y Orenstein [1994] proporcionan una revisión de los sistemas de almacenamiento para bases de datos orientadas a objetos. Jagadish et al. [1994] describen un gestor de almacenamiento para bases de datos en memoria principal.

CAPÍTULO

12 INDEXACIÓN Y ASOCIACIÓN

Muchas consultas hacen referencia sólo a una pequeña parte de los registros de un archivo. Por ejemplo, la pregunta «Buscar todas las cuentas de la sucursal Pamplona» o «Buscar el saldo del número de cuenta C-101» hace referencia solamente a una fracción de los registros de la relación cuenta. No es eficiente para el sistema tener que leer cada registro y comprobar que el campo *nombre-sucursal* contiene el nombre «Pamplona» o el valor C-101 del campo *número-cuenta*. Lo más adecuado sería que el sistema fuese capaz de localizar directamente estos registros. Para facilitar estas formas de acceso se diseñan estructuras adicionales que se asocian con archivos.

12.1. CONCEPTOS BÁSICOS

Un índice para un archivo del sistema funciona como el índice de este libro. Si se va a buscar un tema (especificado por una palabra o una frase) en este libro, se puede buscar en el índice al final del libro, encontrar las páginas en las que aparece y después leer esas páginas para encontrar la información que estamos buscando. Las palabras de índice están ordenadas, lo que hace fácil la búsqueda del término que se esté buscando. Además, el índice es mucho más pequeño que el libro, con lo que se reduce aún más el esfuerzo necesario para encontrar las palabras en cuestión.

Los catálogos de fichas en las bibliotecas funcionan de manera similar (aunque se usan poco). Para encontrar un libro de un autor en particular, se buscaría en el catálogo de autores y una ficha de este catálogo indicaría dónde encontrar el libro. Para ayudarnos en la búsqueda en el catálogo, la biblioteca guardaría en orden alfabético las fichas de los autores con una ficha por cada autor de cada libro.

Los índices de los sistemas de bases de datos juegan el mismo papel que los índices de los libros o los catálogos de fichas de las bibliotecas. Por ejemplo, para recuperar un registro *cuenta* dado su número de cuenta, el sistema de bases de datos buscaría en un índice para encontrar el bloque de disco en que se encuentra el registro correspondiente, y entonces extraería ese bloque de disco para obtener el registro *cuenta*.

Almacenar una lista ordenada de números de cuenta no funcionaría bien en bases de datos muy grandes con millones de cuentas, ya que el propio índice sería muy grande; más aún, incluso al mantener ordenado el índice se reduce el tiempo de búsqueda, encontrar una cuenta puede consumir mucho tiempo. En su lugar se usan técnicas más sofisticadas de indexación. Algunas de estas técnicas se discutirán más adelante.

Hay dos tipos básicos de índices:

- **Índices ordenados.** Estos índices están basados en una disposición ordenada de los valores.
- **Índices asociativos** (*hash indices*). Estos índices están basados en una distribución uniforme de los valores a través de una serie de cajones (*buckets*). El valor asignado a cada cajón está determinado por una función, llamada *función de asociación* (*hash function*).

Se considerarán varias técnicas de indexación y asociación. Ninguna de ellas es la mejor. Sin embargo, cada técnica es la más apropiada para una aplicación específica de bases de datos. Cada técnica debe ser valorada según los siguientes criterios:

- **Tipos de acceso.** Los tipos de acceso que se soportan eficazmente. Estos tipos podrían incluir la búsqueda de registros con un valor concreto en un atributo, o buscar los registros cuyos atributos contengan valores en un rango especificado.
- **Tiempo de acceso.** El tiempo que se tarda en buscar un determinado elemento de datos, o conjunto de elementos, usando la técnica en cuestión.
- **Tiempo de inserción.** El tiempo empleado en insertar un nuevo elemento de datos. Este valor incluye el tiempo utilizado en buscar el lugar apropiado donde insertar el nuevo elemento de datos, así como el tiempo empleado en actualizar la estructura del índice.
- **Tiempo de borrado.** El tiempo empleado en borrar un elemento de datos. Este valor incluye el tiempo utilizado en buscar el elemento a borrar, así como el tiempo empleado en actualizar la estructura del índice.

- **Espacio adicional requerido.** El espacio adicional ocupado por la estructura del índice. Como normalmente la cantidad necesaria de espacio adicional suele ser moderada, es razonable sacrificar el espacio para alcanzar un rendimiento mejor.

A menudo se desea tener más de un índice por archivo. Volviendo al ejemplo de la biblioteca, nos damos cuenta de que la mayoría de las bibliotecas mantienen varios catálogos de fichas: por autor, por materia y por título.

Los atributos o conjunto de atributos usados para buscar en un archivo se llaman **claves de búsqueda**. Hay que observar que esta definición de *clave* difiere de la usada en *clave primaria*, *clave candidata* y *superclave*. Este doble significado de *clave* está (por desgracia) muy extendido en la práctica. Usando nuestro concepto de clave de búsqueda vemos que, si hay varios índices en un archivo, existirán varias claves de búsqueda.

12.2. ÍNDICES ORDENADOS

Para permitir un acceso directo rápido a los registros de un archivo se puede usar una estructura de índice. Cada estructura de índice está asociada con una clave de búsqueda concreta. Al igual que en el catálogo de una biblioteca, un índice almacena de manera ordenada los valores de las claves de búsqueda, y asocia a cada clave los registros que contienen esa clave de búsqueda.

Los registros en el archivo indexado pueden estar a su vez almacenados siguiendo un orden, semejante a como los libros están ordenados en una biblioteca por algún atributo como el número decimal Dewey. Un archivo puede tener varios índices según diferentes claves de búsqueda. Si el archivo que contiene los registros está ordenado secuencialmente, el índice cuya clave de búsqueda especifica el orden secuencial del archivo es el **índice primario**. (El término *índice primario* se emplea algunas veces para hacer alusión a un índice según una clave primaria. Sin embargo, tal uso no es normal y debería evitarse.) Los índices primarios también se llaman **índices con agrupación** (*clustering indices*.) La clave de búsqueda de un índice primario es normalmente la clave primaria, aunque no es así necesariamente. Los índices cuyas claves de búsqueda especifican un orden diferente del orden secuencial del archivo se llaman **índices secundarios** o **índices sin agrupación** (*non clustering indices*).

12.2.1. Índice primario

En este apartado se asume que todos los archivos están ordenados secuencialmente según alguna clave de búsqueda. Estos archivos con índice primario según una clave de búsqueda se llaman **archivos secuenciales indexados**. Representan uno de los esquemas de índices más antiguos usados por los sistemas de bases de datos. Se emplean en aquellas aplicaciones que demandan un procesamiento secuencial del archivo completo así como un acceso directo a sus registros.

En la Figura 12.1 se muestra un archivo secuencial de los registros *cuenta* tomados del ejemplo bancario. En esta figura, los registros están almacenados según el orden de la clave de búsqueda, siendo esta clave *nombre-sucursal*.

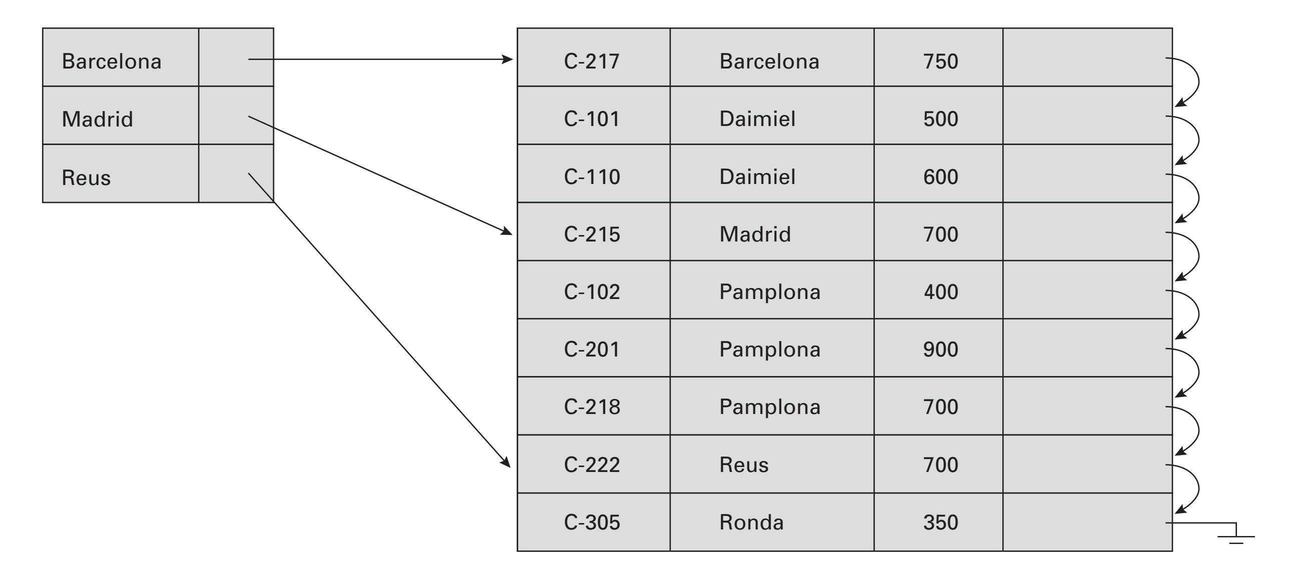


FIGURA 12.1. Archivo secuencial para los registros *cuenta*.

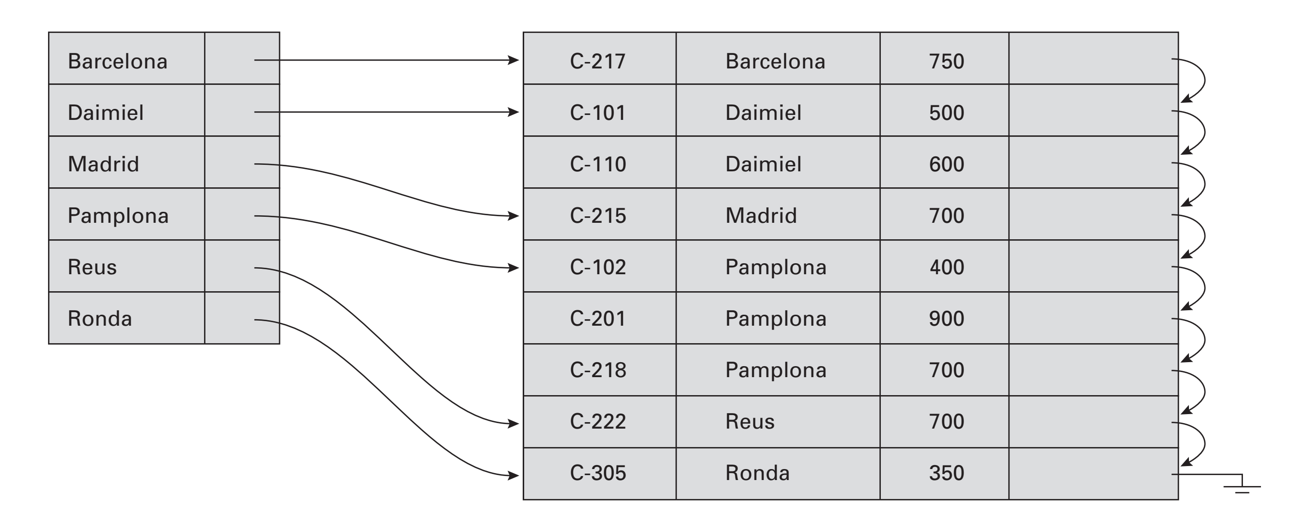


FIGURA 12.2. Índice denso.

12.2.1.1. Índices densos y dispersos

Un **registro índice** o **entrada del índice** consiste en un valor de la clave de búsqueda y punteros a uno o más registros con ese valor de la clave de búsqueda. El puntero a un registro consiste en el identificador de un bloque de disco y un desplazamiento en el bloque de disco para identificar el registro dentro del bloque.

Hay dos clases de índices ordenados que se pueden emplear:

- **Índice denso.** Aparece un registro índice por cada valor de la clave de búsqueda en el archivo. El registro índice contiene el valor de la clave y un puntero al primer registro con ese valor de la clave de búsqueda. El resto de registros con el mismo valor de la clave de búsqueda se almacenan consecutivamente después del primer registro, dado que, ya que el índice es primario, los registros se ordenan sobre la misma clave de búsqueda.

  Las implementaciones de índices densos pueden almacenar una lista de punteros a todos los registros con el mismo valor de la clave de búsqueda; esto no es esencial para los índices primarios.

- **Índice disperso.** Sólo se crea un registro índice para algunos de los valores. Al igual que en los índices densos, cada registro índice contiene un valor de la clave de búsqueda y un puntero al primer registro con ese valor de la clave. Para localizar un registro se busca la entrada del índice con el valor más grande que sea menor o igual que el valor que se está buscando. Se empieza por el registro apuntado por esa entrada del índice y se continúa con los punteros del archivo hasta encontrar el registro deseado.

Las Figuras 12.2 y 12.3 son ejemplos de índices densos y dispersos, respectivamente, para el archivo *cuenta*. Supongamos que se desea buscar los registros de la

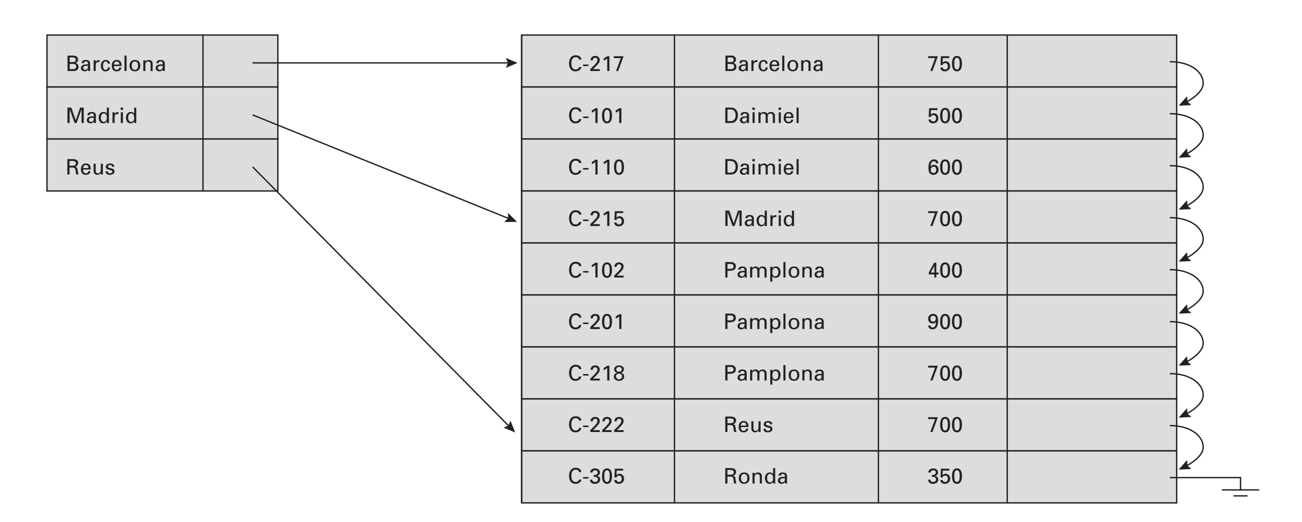


FIGURA 12.3. Índice disperso.

sucursal Pamplona. Mediante el índice denso de la Figura 12.2, se sigue el puntero que va directo al primer registro de Pamplona. Se procesa el registro y se sigue el puntero en ese registro hasta localizar el siguiente registro según el orden de la clave de búsqueda (*nombre-sucursal*). Se continuaría procesando registros hasta encontrar uno cuyo nombre de sucursal fuese distinto de Pamplona. Si se usa un índice disperso (Figura 12.3), no se encontraría entrada del índice para «Pamplona». Como la última entrada (en orden alfabético) antes de «Pamplona» es «Madrid», se sigue ese puntero. Entonces se lee el archivo *cuenta* en orden secuencial hasta encontrar el primer registro Pamplona, y se continúa procesando desde este punto.

Como se ha visto, generalmente es más rápido localizar un registro si se usa un índice denso en vez de un índice disperso. Sin embargo, los índices dispersos tienen algunas ventajas sobre los índices densos, como el utilizar un espacio más reducido y un mantenimiento adicional menor para las inserciones y borrados.

Existe un compromiso que el diseñador del sistema debe mantener entre el tiempo de acceso y el espacio adicional requerido. Aunque la decisión sobre este compromiso depende de la aplicación en particular, un buen compromiso es tener un índice disperso con una entrada del índice por cada bloque. La razón por la cual este diseño alcanza un buen compromiso reside en que el mayor coste de procesar una petición en la base de datos pertenece al tiempo empleado en traer un bloque de disco a la memoria. Una vez traído el bloque, el tiempo en examinar el bloque completo es despreciable. Usando este índice disperso se localiza el bloque que contiene el registro solicitado. De este manera, a menos que el registro esté en un bloque de desbordamiento (véase el Apartado 11.7.1) se minimizan los accesos a bloques mientras mantenemos el tamaño del índice (y así, nuestro espacio adicional requerido) tan pequeño como sea posible.

Para generalizar la técnica anterior hay que tener en cuenta cuando los registros de una clave de búsqueda ocupan varios bloques. Es fácil modificar el esquema para acomodarse a esta situación.

12.2.1.2. Índices multinivel

Incluso si se usan índices dispersos, el propio índice podría ser demasiado grande para un procesamiento eficiente. En la práctica no es excesivo tener un archivo con 100.000 registros, con 10 registros almacenados en cada bloque. Si tenemos un registro índice por cada bloque, el índice tendrá 10.000 registros. Como los registros índices son más pequeños que los registros de datos, podemos suponer que caben 100 registros índices en un bloque. Por tanto, el índice ocuparía 100 bloques. Estos índices de mayor tamaño se almacenan como archivos secuenciales en disco.

Si un índice es lo bastante pequeño como para guardarlo en la memoria principal, el tiempo de búsqueda para encontrar una entrada será breve. Sin embargo, si el índice es tan grande que se debe guardar en disco, buscar una entrada implicará leer varios bloques de disco. Para localizar una entrada en el archivo índice se puede realizar una búsqueda binaria, pero aun así ésta conlleva un gran coste. Si el índice ocupa b bloques, la búsqueda binaria tendrá que leer a lo sumo $\lceil \log_2(b) \rceil$ bloques. ($\lceil x \rceil$ denota al menor entero que es mayor o igual a x; es decir, se redondea hacia abajo.) Para el índice de 100 bloques, la búsqueda binaria necesitará leer siete bloques. En un disco en el que la lectura de un bloque tarda 30 milisegundos, la búsqueda empleará 210 milisegundos, lo que es mucho. Obsérvese que si se están usando bloques de desbordamiento, la búsqueda binaria no sería posible. En ese caso, lo normal es una búsqueda secuencial, y eso requiere leer b bloques, lo que podría consumir incluso más tiempo. Así, el proceso de buscar en un índice grande puede ser muy costoso.

Para resolver este problema se trata el índice como si fuese un archivo secuencial y se construye un índice disperso sobre el índice primario, como se muestra en la Figura 12.4. Para localizar un registro se usa en primer lugar una búsqueda binaria sobre el índice más externo para buscar el registro con el mayor valor de la clave de búsqueda que sea menor o igual al valor deseado. El puntero apunta a un bloque en el índice más interno. Hay que examinar este bloque hasta encontrar el registro con el mayor valor de la clave que sea menor o igual que el valor deseado. El puntero de este registro apunta al bloque del archivo que contiene el registro buscado.

Usando los dos niveles de indexación y con el índice más externo en memoria principal, tenemos que leer un único bloque índice en vez de los siete que se leían con la búsqueda binaria. Si al archivo es extremadamente grande, incluso el índice exterior podría crecer demasiado para caber en la memoria principal. En este caso se podría crear todavía otro nivel más de indexación. De hecho, se podría repetir este proceso tantas veces como fuese necesario. Los índices con dos o más niveles se llaman índices **multinivel**. La búsqueda de registros usando un índice multinivel necesita claramente menos operaciones de E/S que las que se emplean en la búsqueda de registros con la búsqueda binaria. Cada nivel de índice se podría corresponder con una unidad del almacenamiento físico. Así, podríamos tener índices a nivel de pista, cilindro o disco.

Un diccionario normal es un ejemplo de un índice multinivel en el mundo ajeno a las bases de datos. La cabecera de cada página lista la primera palabra en el orden alfabético en esa página. Este índice es multinivel: las palabras en la parte superior de la página del índice del libro forman un índice disperso sobre los contenidos de las páginas del diccionario.

Los índices multinivel están estrechamente relacionados con la estructura de árbol, tales como los árboles binarios usados para la indexación en memoria. Examinaremos esta relación posteriormente en el Apartado 12.3.

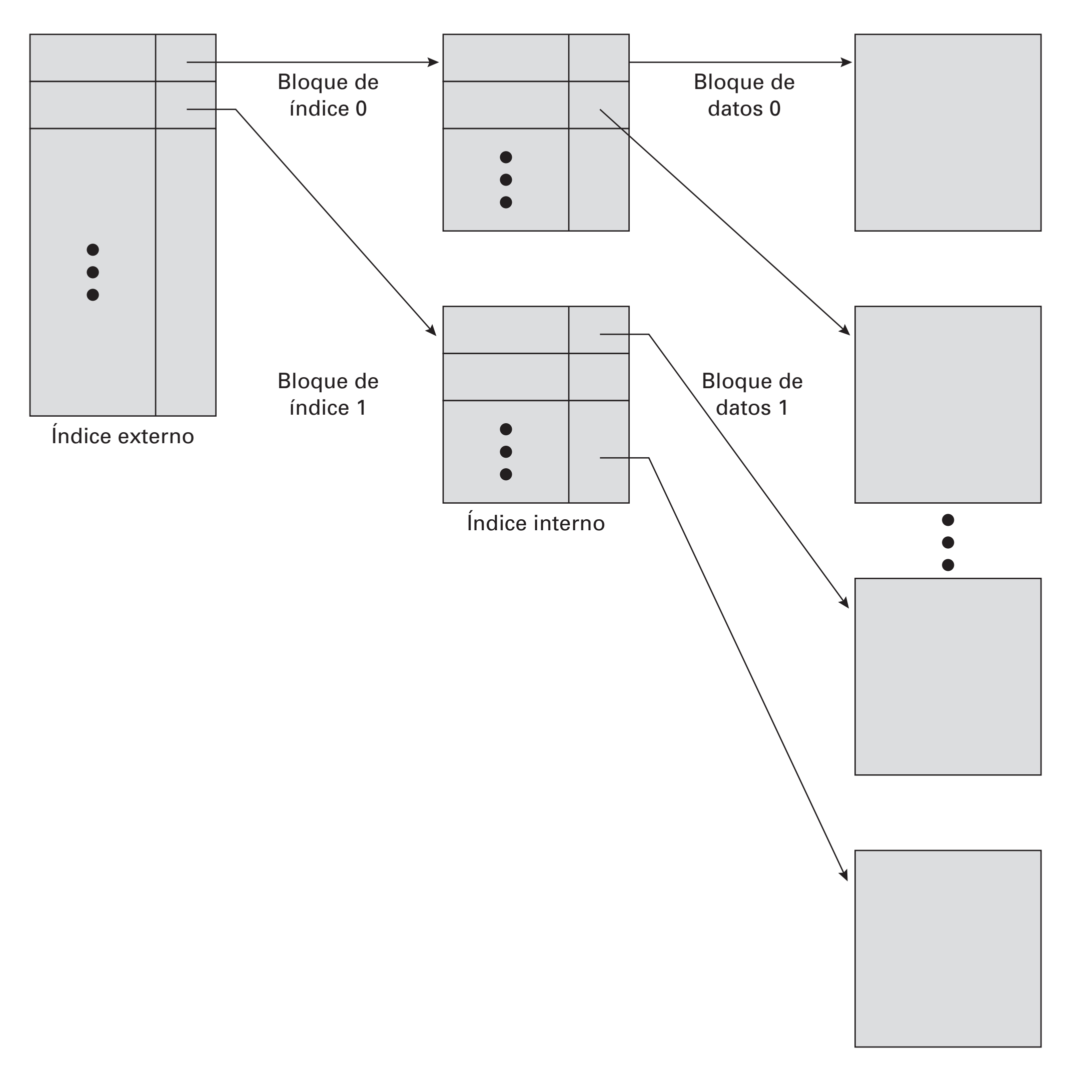


FIGURA 12.4. Índice disperso de dos niveles.

12.2.1.3. Actualización del índice

Sin importar el tipo de índice que se esté usando, los índices se deben actualizar siempre que se inserte o borre un registro del archivo. A continuación se describirán los algoritmos para actualizar índices de un solo nivel.

- **Inserción.** Primero se realiza una búsqueda usando el valor de la clave de búsqueda del registro a insertar. Las acciones que emprende el sistema a continuación dependen de si el índice es denso o disperso.
  - Índices densos:
    1. Si el valor de la clave de búsqueda no aparece en el índice, el sistema inserta en éste un registro índice con el valor de la clave de búsqueda en la posición adecuada.
    2. En caso contrario se emprenden las siguientes acciones:
       - **a.** Si el registro índice almacena punteros a todos los registros con el mismo valor de la clave de búsqueda, el sistema añade un puntero al nuevo registro en el registro índice.
       - **b.** En caso contrario, el registro índice almacena un puntero sólo hacia el primer registro con el valor de la clave de búsqueda. El sistema sitúa el registro insertado después de los otros con los mismos valores de la clave de búsqueda.
  - Índices dispersos: se asume que el índice almacena una entrada por cada bloque. Si el sistema crea un bloque nuevo, inserta el primer valor de la clave de búsqueda (en el orden de la clave de búsqueda) que aparezca en el nuevo bloque del índice. Por otra parte, si el nuevo registro tiene el menor valor de la clave de búsqueda en su bloque, el sistema actualiza la entrada del índice que apunta al bloque; si no,

el sistema no realiza ningún cambio sobre el índice.

- **Borrado.** Para borrar un registro, primero se busca el índice a borrar. De nuevo, las acciones que emprende el sistema a continuación dependen de si el índice es denso o disperso.
  - Índices densos:
    1. Si el registro borrado era el único registro con ese valor de la clave de búsqueda, el sistema borra el registro índice correspondiente del índice.
    2. En caso contrario se emprenden las siguientes acciones:
       a. Si el registro índice almacena punteros a todos los registros con el mismo valor de la clave de búsqueda, el sistema borra del registro índice el puntero al registro borrado.
       b. En caso contrario, el registro índice almacena un puntero sólo al primer registro con el valor de la clave de búsqueda. En este caso, si el registro borrado era el primer registro con el valor de la clave de búsqueda, el sistema actualiza el registro índice para apuntar al siguiente registro.
  - Índices dispersos:
    1. Si el índice no contiene un registro índice con el valor de la clave de búsqueda del registro borrado, no hay que hacer nada.
    2. En caso contrario se emprenden las siguientes acciones:
       a. Si el registro borrado era el único registro con la clave de búsqueda, el sistema reemplaza el registro índice correspondiente con un registro índice para el siguiente valor de la clave de búsqueda (en el orden de la clave de búsqueda). Si el siguiente valor de la clave de búsqueda ya tiene una entrada en el índice, se borra en lugar de reemplazarla.
       b. En caso contrario, si el registro índice para el valor de la clave de búsqueda apunta al registro a borrar, el sistema actualiza el registro índice para que apunte al siguiente registro con el mismo valor de la clave de búsqueda.

Los algoritmos de inserción y borrado para los índices multinivel se extienden de manera sencilla a partir del esquema descrito anteriormente. Al borrar o al insertar se actualiza el índice de nivel más bajo como se describió anteriormente. Por lo que respecta al índice del segundo nivel, el índice de nivel más bajo es simplemente un archivo de registros; así, si hay algún cambio en el índice de nivel más bajo, se tendrá que actualizar el índice del segundo nivel como ya se describió. La misma técnica se aplica al resto de niveles del índice, si los hubiera.

12.2.2. Índices secundarios

Los índices secundarios deben ser densos, con una entrada en el índice por cada valor de la clave de búsqueda, y un puntero a cada registro del archivo. Un índice primario puede ser disperso, almacenando sólo algunos de los valores de la clave de búsqueda, ya que siempre es posible encontrar registros con valores de la clave de búsqueda intermedios mediante un acceso secuencial a parte del archivo, como se describió antes. Si un índice secundario almacena sólo algunos de los valores de la clave de búsqueda, los registros con los valores de la clave de búsqueda intermedios pueden estar en cualquier lugar del archivo y, en general, no se pueden encontrar sin explorar el archivo completo.

Un índice secundario sobre una clave candidata es como un índice denso primario, excepto en que los registros apuntados por los sucesivos valores del índice no están almacenados secuencialmente. Por lo general, los índices secundarios están estructurados de manera diferente a como lo están los índices primarios. Si la clave de búsqueda de un índice primario no es una clave candidata, es suficiente si el valor de cada entrada en el índice apunta al primer registro con ese valor en la clave de búsqueda, ya que los otros registros podrían ser alcanzados por una búsqueda secuencial del archivo.

En cambio, si la clave de búsqueda de un índice secundario no es una clave candidata, no sería suficiente apuntar sólo al primer registro de cada valor de la clave. El resto de registros con el mismo valor de la clave de búsqueda podrían estar en cualquier otro sitio del archivo, ya que los registros están ordenados según la clave de búsqueda del índice primario, en vez de la clave de búsqueda del índice secundario. Por tanto, un índice secundario debe contener punteros a todos los registros.

Se puede usar un nivel adicional de indirección para implementar los índices secundarios sobre claves de búsqueda que no sean claves candidatas. Los punteros en estos índices secundarios no apuntan directamente al archivo. En vez de eso, cada puntero apunta a un cajón que contiene punteros al archivo. En la Figura 12.5 se muestra la estructura del archivo *cuenta*, con un índice secundario que emplea un nivel de indirección adicional, y teniendo como clave de búsqueda el *saldo*.

Siguiendo el orden de un índice primario, una *búsqueda secuencial* es eficiente porque los registros del archivo están guardados físicamente de la misma manera a como está ordenado el índice. Sin embargo, no se puede (salvo en raros casos excepcionales) almacenar el archivo ordenado físicamente por el orden de la clave de búsqueda del índice primario y la clave de búsqueda del índice secundario. Ya que el orden de la cla-

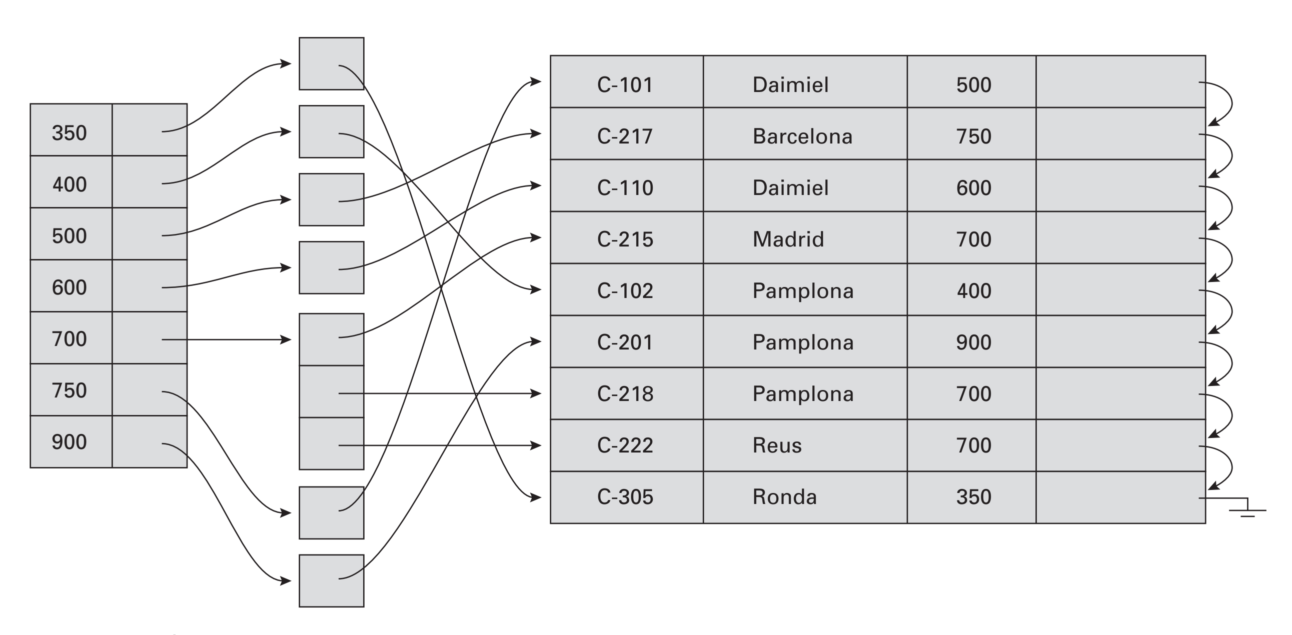


FIGURA 12.5. Índice secundario del archivo *cuenta*, con la clave no candidata *saldo*.

ve secundaria y el orden físico difieren, si se intenta examinar el archivo secuencialmente por el orden de la clave secundaria, es muy probable que la lectura de cada bloque suponga la lectura de un nuevo bloque del disco, lo cual es muy lento.

El procedimiento ya descrito para borrar e insertar se puede aplicar también a los índices secundarios; las acciones a emprender son las descritas para los índices densos que almacenan un puntero a cada registro del archivo. Si un archivo tiene varios índices, siempre que se modifique el archivo, se debe actualizar *cada* uno de ellos.

Los índices secundarios mejoran el rendimiento de las consultas que emplean claves que no son la de búsqueda del índice primario. Sin embargo, implican un tiempo adicional importante al modificar la base de datos. El diseñador de la base de datos debe decidir qué índices secundarios son deseables, según una estimación sobre las frecuencias de las consultas y las modificaciones.

12.3. ARCHIVOS DE ÍNDICES DE ÁRBOL B$^+$

El inconveniente principal de la organización de un archivo secuencial indexado reside en que el rendimiento, tanto para buscar en el índice como para buscar secuencialmente a través de los datos, se degrada según crece el archivo. Aunque esta degradación se puede remediar reorganizando el archivo, el rendimiento de tales reorganizaciones no es deseable.

La estructura de índice de **árbol B$^+$** es la más extendida de las estructuras de índices que mantienen su eficiencia a pesar de la inserción y borrado de datos. Un índice de árbol B$^+$ toma la forma de un **árbol equilibrado** donde los caminos de la raíz a cada hoja del árbol son de la misma longitud. Cada nodo que no es una hoja tiene entre $\lceil n/2 \rceil$ y n hijos, donde n es fijo para cada árbol en particular.

Se verá que la estructura de árbol B$^+$ implica una degradación del rendimiento al insertar y al borrar, además de un espacio extra. Este tiempo adicional es aceptable incluso en archivos con altas frecuencias de modificación, ya que se evita el coste de reorganizar el archivo. Además, puesto que los nodos podrían estar a lo sumo medio llenos (si tienen el mínimo número de hijos) hay algo de espacio desperdiciado. Este gasto de espacio adicional también es aceptable dados los beneficios en el rendimiento aportados por las estructura de árbol B$^+$.

12.3.1. Estructura de árbol B$^+$

Un índice de árbol B$^+$ es un índice multinivel pero con una estructura que difiere del índice multinivel de un archivo secuencial. En la Figura 12.6 se muestra un nodo

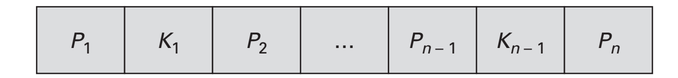


FIGURA 12.6. Nodo típico de un árbol B$^+$.

típico de un árbol B$^+$. Puede contener hasta $n - 1$ claves de búsqueda $K_1, K_2, ..., K_{n-1}$ y n punteros $P_1, P_2, ..., P_n$. Los valores de la clave de búsqueda de un nodo se mantienen ordenados; así, si $i < j$, entonces $K_i < K_j$.

Consideraremos primero la estructura de los nodos hoja. Para $i = 1, 2, ..., n - 1$, el puntero P_i apunta, o bien a un registro del archivo con valor de la clave de búsqueda K_i, o bien a un cajón de punteros, cada uno de los cuales apunta a un registro del archivo con valor de la clave de búsqueda K_i. La estructura cajón se usa solamente si la clave de búsqueda no forma una clave primaria y si el archivo no está ordenado según la clave de búsqueda. El puntero P_n tiene un propósito especial que discutiremos más adelante.

En la Figura 12.7 se muestra un nodo hoja en el árbol B$^+$ del archivo *cuenta*, donde *n* vale tres y la clave de búsqueda es *nombre-sucursal*. Obsérvese que, como el archivo cuenta está ordenado por *nombre-sucursal*, los punteros en el nodo hoja apuntan directamente al archivo.

Ahora que se ha visto la estructura de un nodo hoja, se mostrará cómo los valores de la clave de búsqueda se asignan a nodos concretos. Cada hoja puede guardar hasta $n - 1$ valores. Está permitido que los nodos hojas contengan al menos $\lceil (n - 1)/2 \rceil$ valores. Los rangos de los valores en cada hoja no se solapan. Así, si L_i y L_j son nodos hoja e $i < j$, entonces cada valor de la clave de búsqueda en L_i es menor que cada valor de la clave en L_j. Si el índice de árbol B$^+$ es un índice denso, cada valor de la clave de búsqueda debe aparecer en algún nodo hoja.

Ahora se puede explicar el uso del puntero P_n. Dado que existe un orden lineal en las hojas basado en los valores de la clave de búsqueda que contienen, se usa P_n para encadenar juntos los nodos hojas en el orden de la clave de búsqueda. Esta ordenación permite un procesamiento secuencial eficaz del archivo.

Los nodos internos del árbol B$^+$ forman un índice multinivel (disperso) sobre los nodos hoja. La estructura de los nodos internos es la misma que la de los nodos hoja excepto que todos los punteros son punteros a nodos del árbol. Un nodo interno podría guardar hasta *n* punteros y *debe* guardar al menos $\lceil n/2 \rceil$ punteros. El número de punteros de un nodo se llama *grado de salida* del nodo.

Consideremos un nodo que contiene *m* punteros. Para $i = 2, 3, ..., m - 1$, el puntero P_i apunta al subárbol que contiene los valores de la clave de búsqueda menores que K_i y mayor o igual que K_{i-1}. El puntero P_m apunta a la parte del subárbol que contiene los valores de la clave mayores o iguales que K_{m-1}, y el puntero P_1 apunta a la parte del subárbol que contiene los valores de la clave menores que K_1.

A diferencia de otros nodos internos, el nodo raíz puede tener menos de $\lceil n/2 \rceil$; sin embargo, debe tener al menos dos punteros, a menos que el árbol consista en un solo nodo. Siempre es posible construir un árbol B$^+$, para cualquier *n*, que satisfaga los requisitos anteriores. En la Figura 12.8 se muestra un árbol B$^+$ completo para el archivo *cuenta* ($n = 3$). Por simplicidad se omiten los punteros al propio archivo y los punteros nulos. Como un ejemplo de un árbol B$^+$ en el cual la raíz debe tener menos de $\lceil n/2 \rceil$ valores, en la Figura 12.9 se muestra un árbol B$^+$ para el archivo *cuenta* con $n = 5$.

En todos los ejemplos mostrados de árboles B$^+$, éstos están equilibrados. Es decir, la longitud de cada camino desde la raíz a cada nodo hoja es la misma. Esta propiedad es un requisito de los árboles B$^+$. De hecho, la «B» en árbol B$^+$ proviene del inglés *balanced* (equilibrado). Es esta propiedad de equilibrio de los árboles B$^+$ la que asegura un buen rendimiento para las búsquedas, inserciones y borrados.

12.3.2. Consultas con árboles B$^+$

Considérese ahora cómo procesar consultas usando árboles B$^+$. Supóngase que se desean encontrar todos los registros cuyo valor de la clave de búsqueda sea *V*. La Figura 12.10 muestra el pseudocódigo para hacerlo. Primero se examina el nodo raíz para buscar el menor valor de la clave de búsqueda mayor que *V*. Supóngase que este valor de la clave de búsqueda es K_i. Siguiendo el puntero P_i hasta otro nodo. Si no se encuentra ese valor, entonces $k >= K_{m-1}$, donde *m* es el número de punteros del nodo. Es este caso se sigue P_m hasta otro nodo. En el nodo alcanzado anteriormente se busca de nuevo el

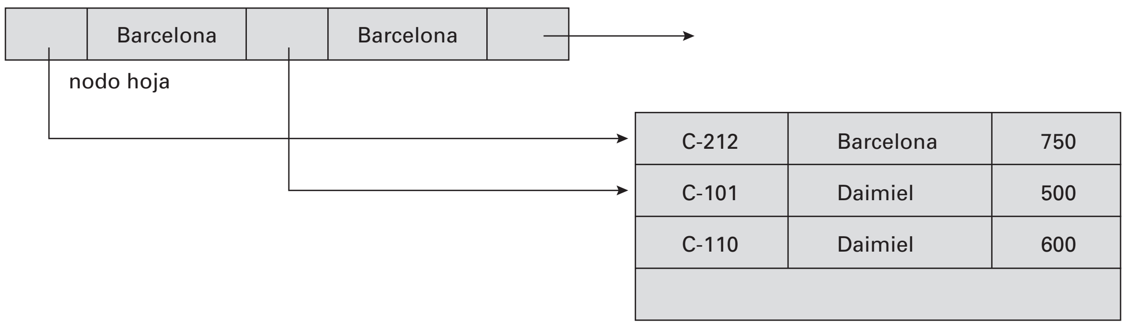


FIGURA 12.7. Nodo hoja para el índice del árbol B$^+$ de *cuenta* ($n = 3$).

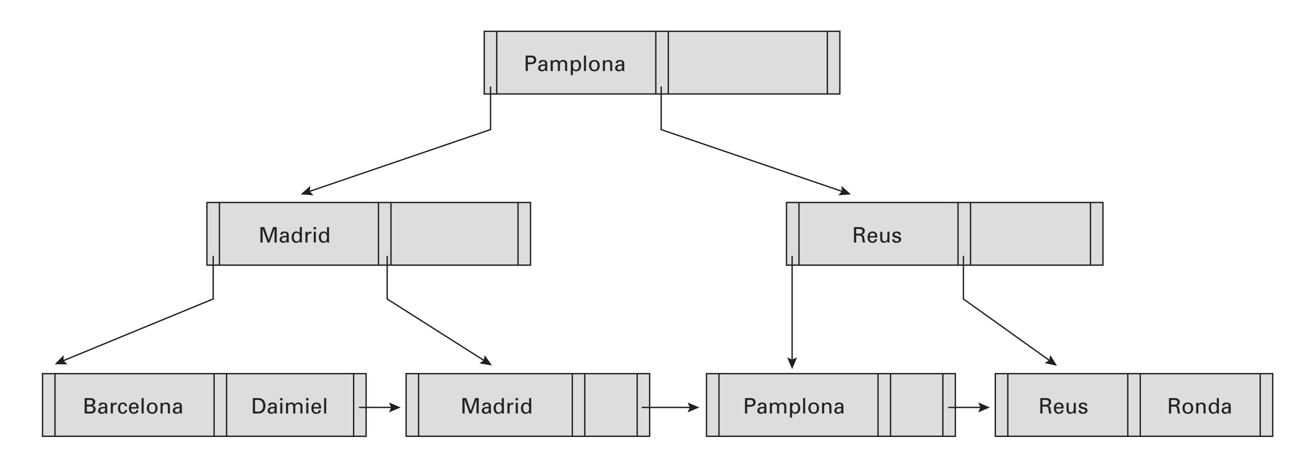


FIGURA 12.8. Árbol B⁺ para el archivo *cuenta* (n = 3).

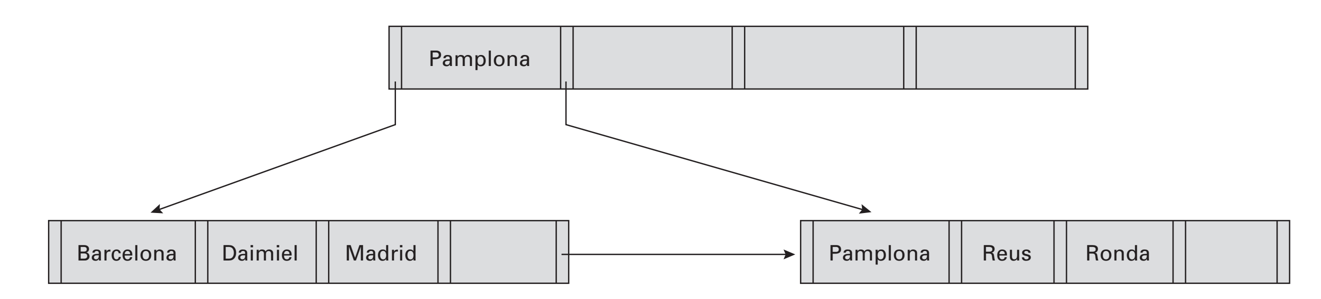


FIGURA 12.9. Árbol B⁺ para el archivo *cuenta* (n = 5).

menor valor de la clave de búsqueda que es mayor que V para seguir el puntero correspondiente. Finalmente se alcanza un nodo hoja. En este nodo hoja, si se encuentra que el valor K_i es igual a V, entonces el puntero P_i nos ha conducido al registro o cajón deseado. Si no se encuentra el valor V en el nodo hoja, no existe ningún registro con el valor clave V.

De esta manera, para procesar una consulta, se tiene que recorrer un camino en el árbol desde la raíz hasta algún nodo hoja. Si hay K valores de la clave de búsqueda en el archivo, este camino no será más largo que $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$.

En la práctica sólo se accede a unos cuantos nodos. Generalmente un nodo se construye para tener el mismo tamaño que un bloque de disco, el cual ocupa normalmente 4 KB. Con una clave de búsqueda del tamaño de 12 bytes y un tamaño del puntero a disco de 8 bytes, n está alrededor de 200. Incluso con una estimación más conservadora de 32 bytes para el tamaño de la clave de búsqueda, n está en torno a 100. Con $n = 100$, si se tienen un millón de valores de la clave de búsqueda en el archivo, una búsqueda necesita solamente $\lceil \log_{50}(1.000.000) \rceil = 4$ accesos a nodos. Por tanto, se necesitan leer a lo sumo cuatro bloques del dis-

```
procedure buscar(valor V)
    set C = nodo raíz
    while C no sea un nodo raíz begin
        Sea Ki = menor valor de la clave de búsqueda, si lo hay, mayor que V
        if no hay tal valor then begin
            Sea m = el número de punteros en el nodo
            Sea C = nodo apuntado por Pm
        end
            else Sea C = el nodo apuntado por Pi
    end
    if hay un valor de clave Ki en C tal que Ki = C
        then el puntero Pi conduce al registro o cajón deseado
        else no existe ningún registro con el valor clave k
end procedure
```

FIGURA 12.10. Consulta con un árbol B⁺.

co para realizar la búsqueda. Normalmente, el nodo raíz del árbol es el más accedido y por ello se guarda en una memoria intermedia; así solamente se necesitan tres o menos lecturas del disco.

Una diferencia importante entre las estructuras de árbol B^+ y los árboles en memoria, tales como los árboles binarios, está en el tamaño de un nodo y, por tanto, la altura del árbol. En un árbol binario, cada nodo es pequeño y tiene a lo sumo dos punteros. En un árbol B^+, cada nodo es grande —normalmente un bloque del disco— y un nodo puede tener un gran número de punteros. Así, los árboles B^+ tienden a ser bajos y anchos, en lugar de los altos y estrechos árboles binarios. En un árbol equilibrado, el camino de una búsqueda puede tener una longitud de $\lceil \log_2 (K) \rceil$, donde K es el número de valores de la clave de búsqueda. Con $K = 1.000.000$, como en el ejemplo anterior, un árbol binario equilibrado necesita alrededor de 20 accesos a nodos. Si cada nodo estuviera en un bloque del disco distinto, serían necesarias 20 lecturas a bloques para procesar la búsqueda, en contraste con las cuatro lecturas del árbol B^+.

12.3.3. Actualizaciones en árboles B^+

El borrado y la inserción son más complicados que las búsquedas, ya que podría ser necesario **dividir** un nodo que resultara demasiado grande como resultado de una inserción, o **fusionar** nodos si un nodo se volviera demasiado pequeño (menor que $\lceil n/2 \rceil$ punteros). Además, cuando se divide un nodo o se fusionan un par de ellos, se debe asegurar que el equilibrio del árbol se mantiene. Para presentar la idea que hay detrás del borrado y la inserción en un árbol B^+, asumiremos que los nodos nunca serán demasiado grandes ni demasiado pequeños. Bajo esta suposición, el borrado y la inserción se realizan como se indica a continuación.

- **Inserción.** Usando la misma técnica que para buscar, se busca un nodo hoja donde tendría que aparecer el valor de la clave de búsqueda. Si el valor de la clave de búsqueda ya aparece en el nodo hoja, se inserta un nuevo registro en el archivo y, si es necesario, un puntero al cajón. Si el valor de la clave de búsqueda no aparece, se inserta el valor en el nodo hoja de tal manera que las claves de búsqueda permanezcan ordenadas. Luego insertamos el nuevo registro en el archivo y, si es necesario, creamos un nuevo cajón con el puntero apropiado.
- **Borrado.** Usando la misma técnica que para buscar, se busca el registro a borrar y se elimina del archivo. Si no hay un cajón asociado con el valor de la clave de búsqueda o si el cajón se queda vacío como resultado del borrado, se borra el valor de la clave de búsqueda del nodo hoja.

Pasamos ahora a considerar un ejemplo en el que se tiene que dividir un nodo. Por ejemplo, queremos insertar un registro en el árbol B^+ de la Figura 12.8, cuyo valor de *nombre-sucursal* es «Cádiz». Usando el algoritmo de búsqueda, «Cádiz» debería aparecer en el nodo que incluye «Barcelona» y «Daimiel». No hay sitio para insertar el valor de la clave de búsqueda «Cádiz». Por tanto, se *divide* el nodo en otros dos nodos. En la Figura 12.11 se muestran los nodos hoja que resultan de insertar «Cádiz» y de dividir el nodo que incluía «Barcelona» y «Daimiel». En general, si tenemos n valores de la clave de búsqueda (los $n - 1$ valores del nodo hoja más el valor a insertar), pondremos $\lceil n/2 \rceil$ en el nodo existente y el resto de valores en el nuevo nodo.

Para dividir un nodo hoja hay que insertar un nuevo nodo hoja en el árbol B^+. En el ejemplo, el nuevo nodo tiene a «Daimiel» como el valor más pequeño de la clave de búsqueda. Luego hay que insertar este valor de la clave de búsqueda en el padre del nodo hoja dividido. En el árbol B^+ de la Figura 12.12 se muestra el resultado de la inserción. El valor «Daimiel» de la clave de búsqueda se ha insertado en el padre. Ha sido posible llevar a cabo esta inserción porque había sitio para añadir un valor de la clave de búsqueda. Si no hubiera sitio, se tendría que dividir el padre. En el peor de los casos, todos los nodos en el camino hacia la raíz se tendrían que dividir. Si la propia raíz se tuviera que dividir, el árbol sería más profundo.

La técnica general para la inserción en un árbol B^+ es determinar el nodo hoja h en el cual realizar la inserción. Si es necesario dividir, se inserta el nuevo nodo dentro del padre del nodo h. Si esta inserción produce otra división, procederíamos recursivamente o bien hasta que una inserción no produzca otra división o bien hasta crear una nueva raíz.

En la Figura 12.13 se bosqueja el algoritmo de inserción en pseudocódigo. En el pseudocódigo $L.K_i$ y $L.P_i$ denotan al i-ésimo valor y el i-ésimo puntero en el nodo L, respectivamente. El pseudocódigo también hace uso de la función *padre*(L) para encontrar el padre del nodo L. Se puede obtener una lista de los nodos en el camino de la raíz a la hoja mientras buscamos el nodo hoja



FIGURA 12.11. División del nodo hoja tras la inserción de «Cádiz».

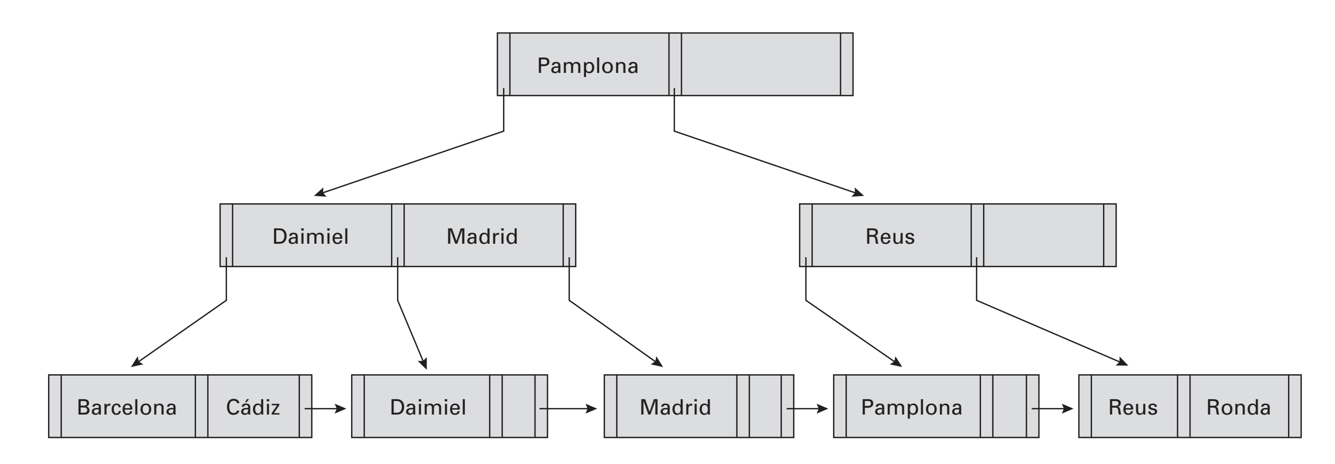


FIGURA 12.12. Inserción de «Cádiz» en el árbol B+ de la Figura 12.8.

para usarlos después a la hora de encontrar eficazmente el padre de cualquier nodo del camino. El pseudocódigo hace referencia a insertar una entrada (V, P) en un nodo. En el caso de nodos hoja, realmente el puntero a una entrada precede al valor de la clave, de tal manera que efectivamente se almacena P en el nodo hoja precediendo a V. Para los nodos internos, se almacena P justo después de V.

```
procedure insertar (valor V, puntero P)
    encontrar el nodo hoja L que debe contener el valor V
    insertar_entrada(L, V, P)
end procedure

procedure insertar_entrada (nodo L, valor V, puntero P)
    if (L tiene espacio para (V, P))
        then    insertar (V, P) en L
        else    begin /* Dividir L */
            Crear el nodo L'
            Sea V' el valor en K1, ...,Kn–1 tal que exactamente ⌈n/2⌉ de los valores L.K1, ...,L.Kn–1, V son menores que V'
            Sea m el menor valor tal que L.Km ≥ V'
            /* Nota: V' tiene que ser L.Km o V */
            if (L es una hoja) then begin
                mover L.Pm, L.Km, ..., L.Pn–1, L.Kn–1 a L'
                if (V < V') then insertar (V, P) en L
                else insertar (V, P) en L'
            end
            else    begin
                if (V = V') /* V es el menor valor que irá en L' */
                    then añadir P, L.Km, ..., L.Pn–1, L.Kn–1, L.Pn a L'
                    else añadir L.Pm, ..., L.Pn–1, L.Kn–1, L.Pn a L'
                borrar L.Km, ..., L.Pn–1, L.Kn–1, L.Pn de L
                if (V < V') then insertar (V, P) en L
                else if (V < V') then insertar (V, P) en L'
                /* El caso V = V' ya se trató anteriormente */
            end
            if (L no es la raíz del árbol)
                then    insertar_entrada(padre(L), V', L');
                else    begin
                    crear un nuevo nodo R con hijos los nodos L y L' y con el único valor V'
                    hacer de R la raíz del árbol
                end
            if (L) es un nodo hoja then begin /* Fijar los siguientes punteros a los hijos */
                hacer L'.Pn = L.Pn;
                hacer L.Pn = L'
            end
        end
end procedure
```

FIGURA 12.13. Inserción de una entrada en un árbol B+.

A continuación se consideran los borrados que provocan que el árbol se quede con muy pocos punteros. Primero se borra «Daimiel» del árbol B^+ de la Figura 12.12. Para ello se localiza la entrada «Daimiel» usando el algoritmo de búsqueda. Cuando se borra la entrada «Daimiel» de su nodo hoja, la hoja se queda vacía. Ya que en el ejemplo $n = 3$ y $0 < \lceil (n-1)/2 \rceil$, este nodo se debe borrar del árbol B^+. Para borrar un nodo hoja se tiene que borrar el puntero que le llega desde su padre. En el ejemplo, este borrado deja al nodo padre, el cual contenía tres punteros, con sólo dos punteros. Ya que $2 >= \lceil n/2 \rceil$, el nodo es todavía lo suficientemente grande y la operación de borrado se completa. El árbol B^+ resultante se muestra en la Figura 12.14.

Cuando un borrado se hace sobre el padre de un nodo hoja, el propio nodo padre podría quedar demasiado pequeño. Esto es exactamente los que ocurre si se borra «Pamplona» del árbol B^+ de la Figura 12.14. El borrado de la entrada Pamplona provoca que un nodo hoja se quede vacío. Cuando se borra el puntero a este nodo en su padre, el padre sólo se queda con un puntero. Así, $n = 3$, $\lceil n/2 \rceil = 2$ y queda tan sólo un puntero, que es demasiado poco. Sin embargo, ya que el nodo padre contiene información útil, no podemos simplemente borrarlo. En vez de eso, se busca el nodo hermano (el nodo interno que contiene al menos una clave de búsqueda, Madrid). Este nodo hermano tiene sitio para colocar la información contenida en el nodo que quedó demasiado pequeño, así que se fusionan estos nodos, de tal manera que el nodo hermano ahora contiene las claves «Madrid» y «Reus». El otro nodo (el nodo que contenía solamente la clave de búsqueda «Reus») ahora contiene información redundante y se puede borrar desde su padre (el cual resulta ser la raíz del ejemplo). En la Figura 12.15 se muestra el resultado. Hay que observar que la raíz tiene solamente un puntero hijo como resultado del borrado, así que éste se borra y el hijo solitario se convierte en la nueva raíz. De esta manera la profundidad del árbol ha disminuido en una unidad.

No siempre es posible fusionar nodos. Como ejemplo se borrará «Pamplona» del árbol B^+ de la Figura 11.11. En este ejemplo, la entrada «Daimiel» es todavía parte del árbol. Una vez más, el nodo hoja que contiene «Pamplona» se queda vacío. El padre del nodo hoja se queda también demasiado pequeño (únicamente con un puntero). De cualquier modo, en este ejemplo, el nodo hermano contiene ya el máximo número de punteros: tres. Así, no puede acomodar a un puntero adicional. La solución en este caso es **redistribuir** los punteros de tal manera que cada hermano tenga dos punteros. El resultado se muestra en la Figura 12.16. Obsérvese que la redistribución de los valores necesita de un cambio en el valor de la clave de búsqueda en el padre de los dos hermanos.

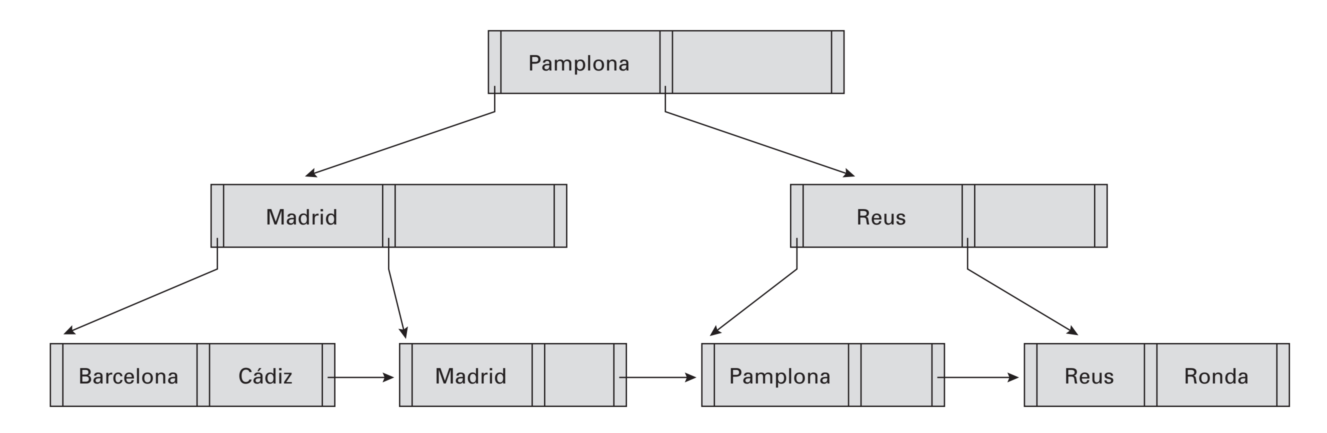


FIGURA 12.14. Borrado de «Daimiel» del árbol B^+ de la Figura 12.12.

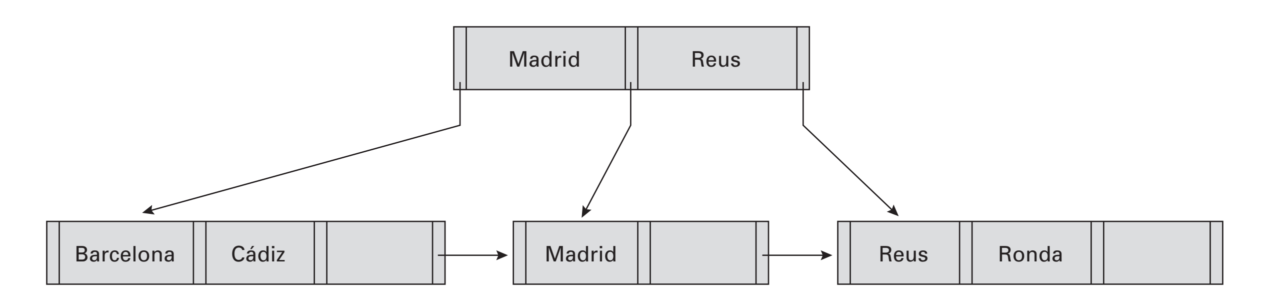


FIGURA 12.15. Borrado de «Pamplona» del árbol B^+ de la Figura 12.14.

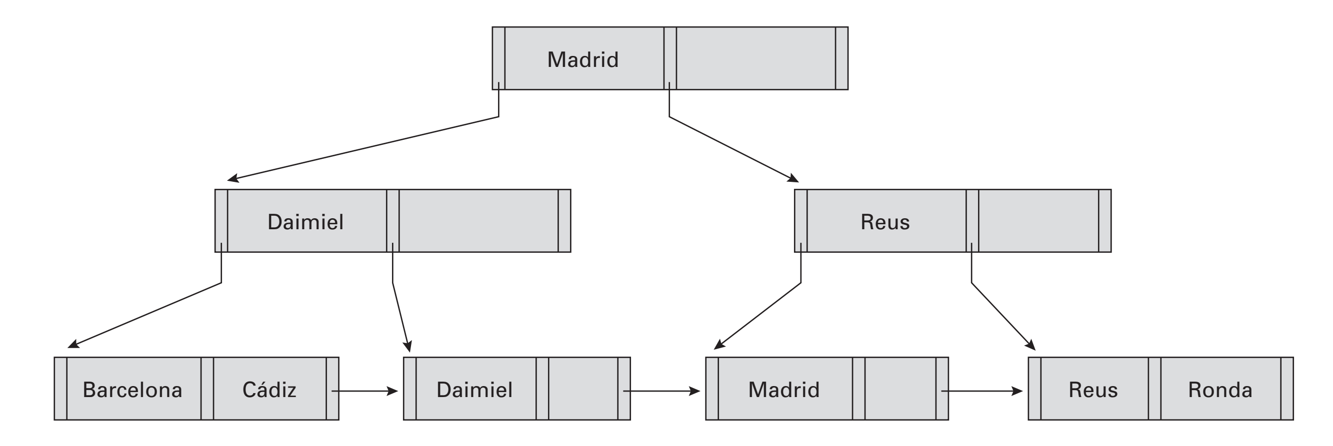


FIGURA 12.16. Borrado de «Pamplona» del árbol B$^+$ de la Figura 12.12.

En general, para borrar un valor en un árbol B$^+$ se realiza una búsqueda según el valor y se borra. Si el nodo es demasiado pequeño, se borra desde su padre. Este borrado se realiza como una aplicación recursiva del algoritmo de borrado hasta que se alcanza la raíz, un nodo padre queda lleno de manera adecuada después de borrar, o hasta aplicar una redistribución.

En la Figura 12.17 se describe el pseudocódigo para el borrado en un árbol B$^+$. El procedimiento intercambiar_variables(L, L') simplemente cambia de lugar los valores (punteros) de las variables L y L'; este cambio no afecta al árbol en sí mismo. El pseudocódigo utiliza la condición «muy pocos valores/punteros». Para nodos internos, este criterio quiere decir menos que $\lceil n/2 \rceil$ punteros; para nodos hoja, quiere decir menos que $\lceil (n-1)/2 \rceil$ valores. El pseudocódigo realiza la redistribución tomando prestada una sola entrada desde un nodo adyacente. También se puede redistribuir mediante la distribución equitativa de entradas entre dos nodos. El pseudocódigo hace referencia al borrado de una entrada (V, P) desde un nodo. En el caso de los nodos hoja, el puntero a una entrada realmente precede al valor de la clave; así, el puntero P precede al valor de la clave V. Para nodos internos, P sigue al valor de la clave V.

Es interesante señalar que como resultado de un borrado, un valor de la clave de un nodo interno del árbol B$^+$ puede que no esté en ninguna hoja del árbol.

Aunque las operaciones inserción y borrado en árboles B$^+$ son complicadas, requieren relativamente pocas operaciones, lo que es un beneficio importante dado el coste de las operaciones E/S. Se puede demostrar que el número de operaciones E/S necesarias para una inserción o borrado es, en el peor de los casos, proporcional a $\log_{\lceil n/2 \rceil}(K)$, donde n es el número máximo de punteros en un nodo y K es el número de valores de la clave de búsqueda. En otras palabras, el coste de las operaciones inserción y borrado es proporcional a la altura del árbol B$^+$ y es por lo tanto bajo. Debido a la velocidad de las operaciones en los árboles B$^+$, estas estructuras de índice se usan frecuentemente al implementar las bases de datos.

12.3.4. Organización de archivos con árboles B$^+$

Como se mencionó en el Apartado 12.3, el inconveniente de la organización de archivos secuenciales de índices es la degradación del rendimiento según crece el archivo: con el crecimiento, un porcentaje mayor de registros índice y registros reales se desaprovechan y se almacenan en bloques de desbordamiento. Se resuelve la degradación de las búsquedas en el índice mediante el uso de índices de árbol B$^+$ en el archivo. También se soluciona el problema de la degradación al almacenar los registros reales utilizando el nivel de hoja del árbol B$^+$ para almacenar los registros reales en los bloques. En estas estructuras, la estructura del árbol B$^+$ se usa no sólo como un índice, sino también como un organizador de los registros dentro del archivo. En la **organización de archivo con árboles B$^+$**, los nodos hoja del árbol almacenan registros, en lugar de almacenar punteros a registros. En la Figura 12.18 se muestra un ejemplo de la organización de un archivo con un árbol B$^+$. Ya que los registros son normalmente más grandes que los punteros, el número máximo de registros que se pueden almacenar en un nodo hoja es menor que el número de punteros en un nodo interno. Sin embargo, todavía se requiere que los nodos hoja estén llenos al menos hasta la mitad.

La inserción y borrado de registros en una organización de archivo con árboles B$^+$ se trata del mismo modo que la inserción y borrado de entradas en un índice de árbol B$^+$. Cuando se inserta un registro con un valor de clave v, se localiza el bloque que debería contener ese registro mediante la búsqueda en el árbol B$^+$ de la mayor clave que sea $\leq v$. Si el bloque localizado tiene bastante espacio libre para el registro, se almacena el registro en el bloque. De no ser así, como en una inserción en un árbol B$^+$, se divide el bloque en dos y se redistribuyen sus registros (en el orden de la clave

```
procedure borrar (valor V, puntero P)
    encontrar el nodo hoja L que contiene (V, P)
    borrar_entrada(L, V, P)
end procedure

procedure borrar_entrada (nodo L, valor V, puntero P)
    borrar (V, P) de L
    if (L es la raíz and L tiene sólo un hijo)
    then hacer del hijo de L la nueva raíz del árbol y borrar L
    else if (L tiene muy pocos valores/punteros) then begin
            Sea L′ el anterior o el siguiente hijo de padre(L)
            Sea V′ el valor enter los punteros L y L′ en padre(L)
            if las entradas en L y L′ caben en un solo nodo)
                then begin /* Fundir nodos */
                    if (L es un predecesor de L′) then intercambiar_variables(L, L′)
                    if (L no es una hoja)
                        then concatenar V′ y todos los punteros y valores de L a L′
                        else concatenar todos los pares (K_j, P_j) de L a L′; hacer L′.P_n = L.P_n

                    borrar_entrada(padre(L), V′, L); borrar el nodo L
                end
            else begin /* Redistribución : prestar una entrada de L′*/
                if (L′ es un predecesor de L) then begin
                    if (L es un nodo interno) then begin
                        sea m tal que L′.P_m es el último puntero en L′
                        suprimir (L′.K_m–1, L′.P_m) de L′
                        insertar (L′.P_m, V′) como el primer puntero y valor en L, desplazando otros punteros y valores a la derecha
                        reemplazar V′ en padre(L) por L′.K_m–1
                    end
                    else begin
                        sea m tal que (L′.P_m, L′.K_m) es el último par puntero/valor en L′
                        suprimir (L′.P_m, L′.K_m) de L′
                        insertar (L′.P_m, L′.K_m) como el primer puntero y valor en L, desplazando otros punteros y valores a la derecha
                        reemplazar V′ en padre(L) por L′.K_m
                    end
                end
                else ... caso simétrico al then ...
            end
        end
end procedure
```

FIGURA 12.17. Borrado de elementos de un árbol B⁺.

del árbol B⁺) creando espacio para el nuevo registro. Esta división se propaga hacia arriba en el árbol B⁺ de la manera usual. Cuando se borra un registro, primero se elimina del bloque que lo contiene. Si como resultado un bloque *B* llega a ocupar menos que la mitad, los registros en *B* se redistribuyen con los registros en un bloque *B*′ adyacente. Si se asume que los registros son de tamaño fijo, cada bloque contendrá por lo menos la

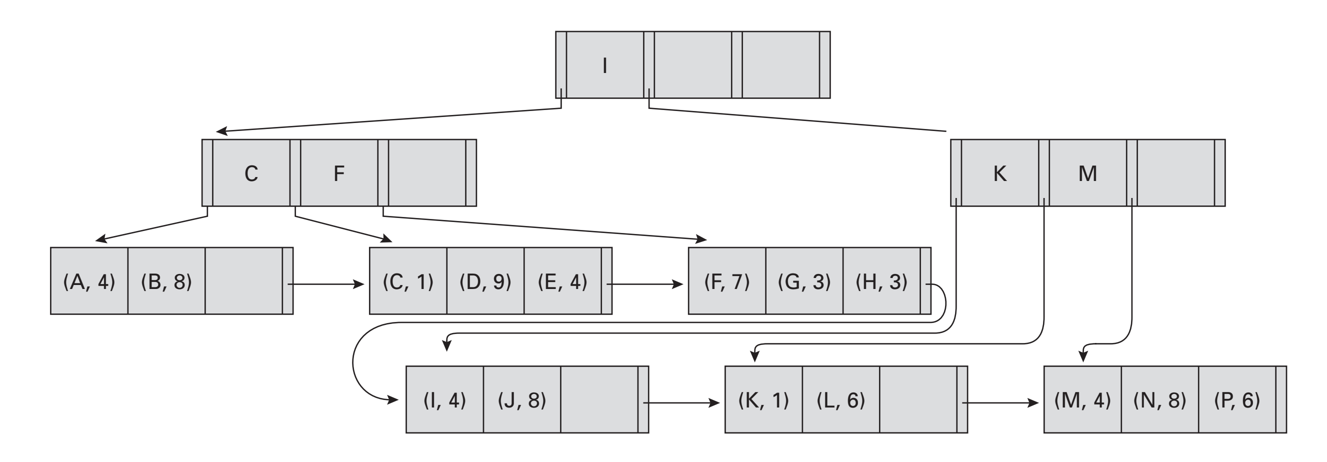


FIGURA 12.18. Organización de archivos con árboles B⁺.

mitad de los registros que pueda contener como máximo. Los nodos internos del árbol B$^+$ se actualizan por tanto de la manera habitual.

Cuando un árbol B$^+$ se utiliza para la organización de un archivo, la utilización del espacio es particularmente importante, ya que el espacio ocupado por los registros es mucho mayor que el espacio ocupado por las claves y punteros. Se puede mejorar la utilización del espacio en un árbol B$^+$ implicando a más nodos hermanos en la redistribución durante las divisiones y fusiones. La técnica es aplicable a los nodos hoja y nodos internos y funciona como sigue.

Durante la inserción, si un nodo está lleno se intenta redistribuir algunas de sus entradas en uno de los nodos adyacentes para hacer sitio a una nueva entrada. Si este intento falla porque los nodos adyacentes están llenos, se divide el nodo y las entradas entre uno de los nodos adyacentes y los dos nodos que se obtienen al dividir el nodo original. Puesto que los tres nodos juntos contienen un registro más que puede encajar en dos nodos, cada nodo estará lleno aproximadamente hasta sus dos terceras partes. Para ser más precisos, cada nodo tendrá por lo menos $\lfloor 2n/3 \rfloor$ entradas, donde n es el número máximo de entradas que puede tener un nodo ($\lfloor n \rfloor$ denota el mayor entero menor o igual que x; es decir, eliminamos la parte fraccionaria si la hay).

Durante el borrado de un registro, si la ocupación de un nodo está por debajo de $\lfloor 2n/3 \rfloor$, se intentará tomar prestada una entrada desde uno de sus nodos hermanos. Si ambos nodos hermanos tienen $\lfloor 2n/3 \rfloor$ registros, en lugar de tomar prestada una entrada, se redistribuyen las entradas en el nodo y en los dos nodos hermanos uniformemente entre dos de los nodos y se borra el tercer nodo. Se puede usar este enfoque porque el número total de entradas es $3 \lfloor 2n/3 \rfloor - 1$, lo cual es menor que $2n$. Utilizando tres nodos adyacentes para la redistribución se garantiza que cada nodo pueda tener $\lfloor 3n/4 \rfloor$ entradas. En general, si hay m nodos ($m - 1$ hermanos) implicados en la redistribución se garantiza que cada nodo pueda contener al menos $\lfloor (m - 1)n/m \rfloor$ entradas. Sin embargo, el costo de la actualización se vuelve mayor según haya más nodos hermanos involucrados en la redistribución.

12.4. ARCHIVOS CON ÍNDICES DE ÁRBOL B

Los *índices de árbol B* son similares a los índices de árbol B$^+$. La diferencia principal entre los dos enfoques es que un árbol B elimina el almacenamiento redundante de los valores de la clave búsqueda. En el árbol B$^+$ de la Figura 12.12, las claves de búsqueda «Daimiel», «Madrid», «Reus» y «Pamplona» aparecen dos veces. Cada valor de clave de búsqueda aparece en algún nodo hoja; algunos se repiten en nodos internos.

Un árbol B permite que los valores de la clave de búsqueda aparezcan solamente una vez. En la Figura 12.19 se muestra un árbol B que representa las mismas claves de búsqueda que el árbol B$^+$ de la Figura 12.12. Ya que las claves de búsqueda no están repetidas en el árbol B, sería posible almacenar el índice empleando menos nodos del árbol que con el correspondiente índice de árbol B$^+$. Sin embargo, puesto que las claves de búsqueda que aparecen en los nodos internos no aparecen en ninguna otra parte del árbol B, nos vemos obligados a incluir un campo adicional para un puntero por cada clave de búsqueda de un nodo interno. Estos punteros adicionales apuntan a registros del archivo o a los cajones de la clave de búsqueda asociada.

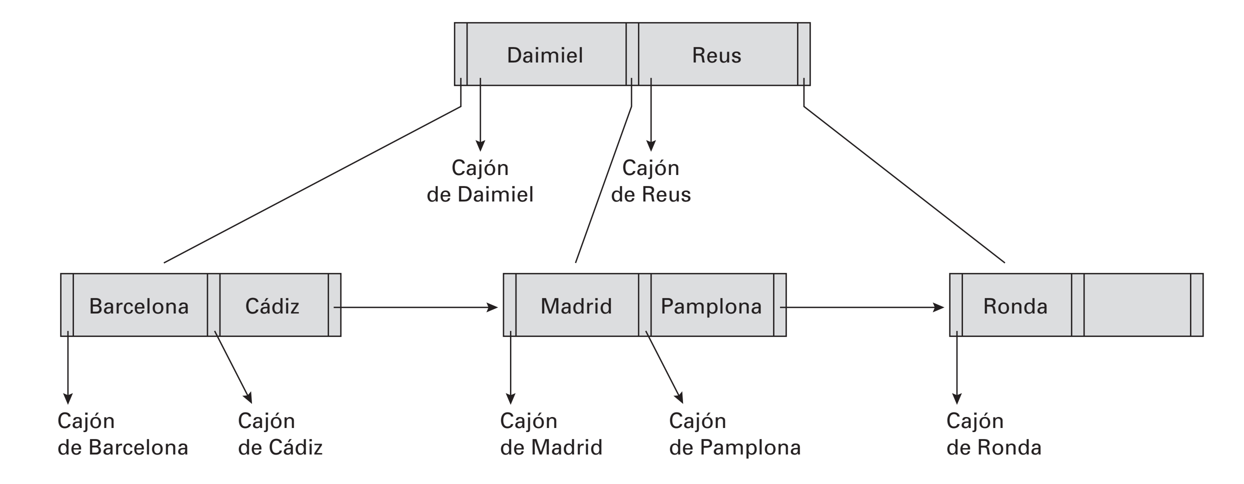


FIGURA 12.19. Árbol B equivalente al árbol B$^+$ de la Figura 12.12.

Un nodo hoja generalizado de un árbol B aparece en la Figura 12.20a; un nodo interno aparece en la Figura 12.20b. Los nodos hoja son como en los árboles B^+. En los nodos internos los punteros P_i son los punteros del árbol que se utilizan también para los árboles B^+, mientras que los punteros B_i en los nodos internos son punteros a cajones o registros del archivo. En la figura del árbol B generalizado hay $n – 1$ claves en el nodo hoja, mientras que hay $m – 1$ claves en el nodo interno. Esta discrepancia ocurre porque los nodos internos deben incluir los punteros B_i, y de esta manera se reduce el número de claves de búsqueda que pueden contener estos nodos. Claramente, $m < n$, pero la relación exacta entre m y n depende del tamaño relativo de las claves de búsqueda y de los punteros.

El número de nodos accedidos en una búsqueda en un árbol B depende de dónde esté situada la clave de búsqueda. Una búsqueda en un árbol B^+ requiere atravesar un camino desde la raíz del árbol hasta algún nodo hoja. En cambio, algunas veces es posible encontrar en un árbol B el valor deseado antes de alcanzar el nodo hoja. Sin embargo, hay que realizar aproximadamente n accesos según cuántas claves haya almacenadas tanto en el nivel de hoja de un árbol B como en los niveles internos de hoja y, dado que n es normalmente grande, la probabilidad de encontrar ciertos valores pronto es relativamente pequeña. Por otra parte, el hecho de que menos claves de búsqueda aparezcan en los nodos internos del árbol B, comparado con los árboles B^+, implica que un árbol B tiene un grado de salida menor y, por lo tanto, puede que tenga una profundidad mayor que el correspondiente al árbol B^+. Así, la búsqueda en un árbol B es más rápida para algunas claves de búsqueda pero más lenta para otras, aunque en general, el tiempo de la búsqueda es todavía proporcional al logaritmo del número de claves de búsqueda.

El borrado en un árbol B es más complicado. En un árbol B^+ la entrada borrada siempre aparece en una hoja. En un árbol B, la entrada borrada podría aparecer en un nodo interno. El valor apropiado a colocar en su lugar se debe elegir del subárbol del nodo que contiene la entrada borrada. Concretamente, si se borra la clave de búsqueda K_i, la clave de búsqueda más pequeña que aparezca en el subárbol del puntero $P_i + 1$ se debe trasladar al campo ocupado anteriormente por K_i. Será necesario tomar otras medidas si ahora el nodo hoja tuviera pocas entradas. Por el contrario, la inserción en un árbol B es sólo un poco más complicada que la inserción en un árbol B^+.

Las ventajas de espacio que tienen los árboles B son escasas para índices grandes y normalmente no son de mayor importancia los inconvenientes que hemos advertido. De esta manera, muchos implementadores de sistemas de bases de datos aprovechan la sencillez estructural de un árbol B^+. Los detalles de los algoritmos de inserción y borrado para árboles B se estudian en los ejercicios.

12.5. ASOCIACIÓN ESTÁTICA

Un inconveniente de la organización de archivos secuenciales es que hay que acceder a una estructura de índices para localizar los datos o utilizar una búsqueda binaria y, como resultado, más operaciones de E/S. La organización de archivos basada en la técnica de **asociación** (*hashing*) permite evitar el acceso a la estructura de índice. La asociación también proporciona una forma de construir índices. Se estudiarán las organizaciones de archivos e índices basados en asociación en los próximos apartados.

12.5.1. Organización de archivos por asociación

En una **organización de archivos por asociación** se obtiene la dirección del bloque de disco que contiene el registro deseado mediante el cálculo directo de una función sobre el valor de la clave búsqueda del registro. En nuestra descripción de asociación, utilizaremos el término **cajón** (*bucket*) para indicar una unidad de almacenamiento que puede guardar uno o más registros. Un

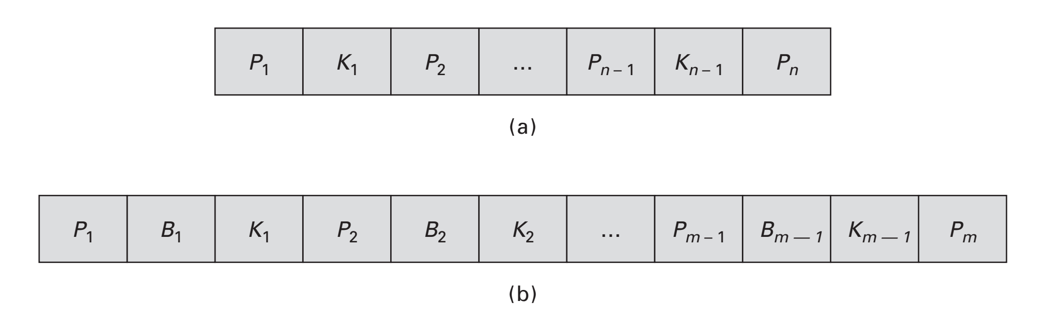


FIGURA 12.20. Nodos típicos de un árbol B. (a) Nodo hoja. (b) Nodo interno.

cajón es normalmente un bloque de disco, aunque también se podría elegir más pequeño o más grande que un bloque de disco.

Formalmente, sea K el conjunto de todos los valores de clave de búsqueda y sea B el conjunto de todas las direcciones de cajón. Una **función de asociación** h es una función de K a B. Sea h una función asociación.

Para insertar un registro con clave de búsqueda K_i, calcularemos $h(K_i)$, lo que proporciona la dirección del cajón para ese registro. Se supone que por ahora hay espacio en el cajón para almacenar el registro. Entonces, el registro se almacena en este cajón.

Para realizar una búsqueda con el valor K_i de la clave de búsqueda, simplemente se calcula $h(K_i)$ y luego se busca el cajón con esa dirección. Supongamos que dos claves de búsqueda, K_5 y K_7, tienen el mismo valor de asociación; esto es, $h(K_5) = h(K_7)$. Si se realiza una búsqueda en K_5, el cajón $h(K_5)$ contendrá registros con valores de la clave de búsqueda K_5 y registros con valores de la clave de búsqueda K_7. Así, hay que comprobar el valor de clave de búsqueda de cada registro en el cajón para verificar que el registro es el que queremos.

El borrado es igual de sencillo. Si el valor de clave de búsqueda del registro a borrar es K_i, se calcula $h(K_i)$, después se busca el correspondiente cajón para ese registro y se borra el registro del cajón.

12.5.1.1. Funciones de asociación

La peor función posible de asociación asigna todos los valores de la clave de búsqueda al mismo cajón. Tal función no es deseable, ya que todos los registros tienen que guardarse en el mismo cajón. Una búsqueda tiene que examinar cada registro hasta encontrar el deseado. Una función de asociación ideal distribuye las claves almacenadas uniformemente a través de los cajones para que cada uno de ellos tenga el mismo número de registros.

Puesto que no se sabe durante la etapa de diseño qué valores de la clave de búsqueda se almacenarán en el archivo, se pretende elegir una función de asociación que asigne los valores de las claves de búsqueda a los cajones de manera que se cumpla lo siguiente:

- Distribución *uniforme*. Esto es, cada cajón tiene asignado el mismo número de valores de la clave de búsqueda dentro del conjunto de *todos* los valores posibles de la clave de búsqueda.
- Distribución *aleatoria*. Esto es, en el caso promedio, cada cajón tendrá casi el mismo número de valores asignados a él, sin tener en cuenta la distribución actual de los valores de la clave de búsqueda. Para ser más exactos, el valor de asociación no será correlativo a ninguna orden exterior visible en los valores de la clave de búsqueda, como por ejemplo el orden alfabético o el orden determinado por la longitud de las claves de búsqueda; la función de asociación tendrá que parecer ser aleatoria.

Como una ilustración de estos principios, vamos a escoger una función de asociación para el archivo *cuenta* utilizando la clave búsqueda *nombre-sucursal*. La función de asociación que se escoja debe tener las propiedades deseadas no sólo en el ejemplo del archivo *cuenta* que se ha estado utilizando, sino también en el archivo *cuenta* de tamaño real para un gran banco con muchas sucursales.

Supóngase que se decide tener 26 cajones y se define una función de asociación que asigna a los nombres que empiezan con la letra *i*-ésima del alfabeto el *i*-ésimo cajón. Esta función de asociación tiene la virtud de la simplicidad, pero no logra proporcionar una distribución uniforme, ya que se espera tener más nombres de sucursales que comienzan con letras como «B» y «R» que con «Q» y «X», por citar un ejemplo.

Ahora supóngase que se quiere una función de asociación en la clave de búsqueda *saldo*. Supóngase que el saldo mínimo es 1 y el saldo máximo es 100.000, se utiliza una función de asociación que divide el valor en 10 rangos, 1-10.000, 10.001-20.000, y así sucesivamente. La distribución de los valores de la clave de búsqueda es uniforme (ya que cada cajón tiene el mismo número de valores del *saldo* diferente), pero no es aleatorio. Los registros con saldos entre 1 y 10.000 son más comunes que los registros con saldos entre 90.001 y 100.000. Como resultado, la distribución de los registros no es uniforme: algunos cajones reciben más registros que otros. Si la función tiene una distribución aleatoria, incluso si hubiera estas correlaciones en las claves de búsqueda, la aleatoriedad de la distribución hará que todos los cajones tengan más o menos el mismo número de registros, siempre que cada clave de búsqueda aparezca sólo en una pequeña fracción de registros. (Si una única clave de búsqueda aparece en una gran fracción de registros, el cajón que la contiene probablemente tenga más registros que otros cajones, independientemente de la función de asociación usada.)

Las funciones de asociación típicas realizan cálculos sobre la representación binaria interna de la máquina para los caracteres de la clave de búsqueda. Una función de asociación simple de este tipo, en primer lugar, calcula la suma de las representaciones binarias de los caracteres de una clave y, posteriormente, devuelve la suma módulo al número de cajones. En la Figura 12.21 se muestra la aplicación de este esquema, utilizando 10 cajones, para el archivo *cuenta*, bajo la suposición de que la letra *i*-ésima del alfabeto está representada por el número entero *i*.

Las funciones de asociación requieren un diseño cuidadoso. Una mala función de asociación podría provocar que una búsqueda tome un tiempo proporcional al número de claves de búsqueda en el archivo. Una función bien diseñada en un caso medio de búsqueda toma un tiempo constante (pequeño), independiente del número de claves búsqueda en el archivo.

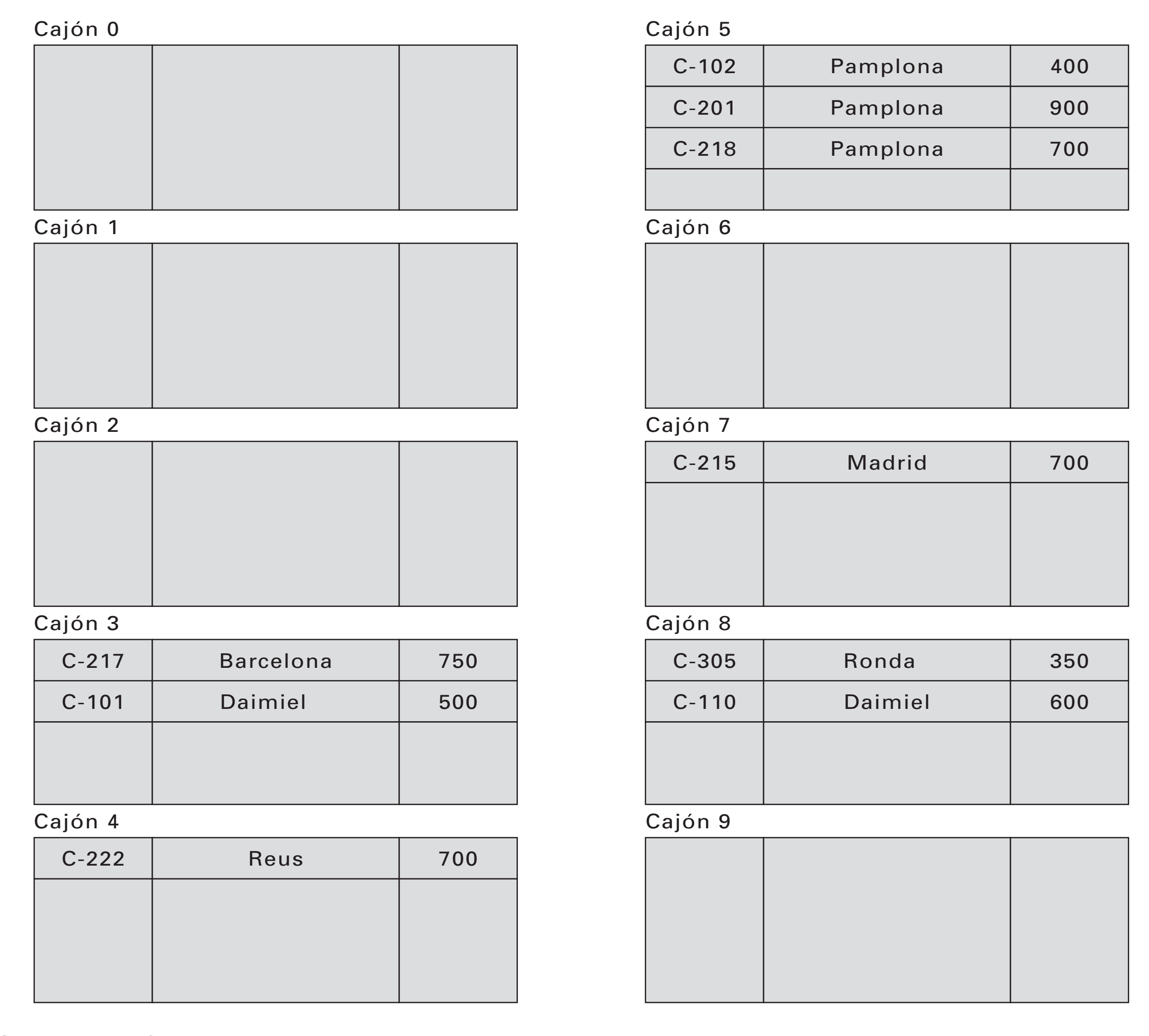


FIGURA 12.21. Organización asociativa del archivo *cuenta* utilizando *nombre-sucursal* como clave.

12.5.1.2. Gestión de desbordamientos de cajones

Hasta ahora se ha asumido que cuando se inserta un registro, el cajón al que se asigna tiene espacio para almacenar el registro. Si el cajón no tiene suficiente espacio, sucede lo que se denomina **desbordamiento de cajones**. Los desbordamientos de cajones pueden ocurrir por varias razones:

- **Cajones insuficientes.** El número de cajones, que se denota con n_B, debe ser escogido tal que $n_C > n_r / f_r$, donde n_r denota el número total de registros que serán almacenados, y f_r denota el número de registros que se colocarán en un cajón. Esta designación, por supuesto, está bajo la suposición de que se conoce el número total de registros cuando se define la función de asociación.
- **Atasco.** Algunos cajones tienen asignados más registros que otros, por lo que un cajón se podría desbordar incluso cuando otros cajones tienen todavía espacio. Esta situación se denomina *atasco* de cajones. El atasco puede ocurrir por dos razones:
  1. Varios registros podrían tener la misma clave de búsqueda.
  2. La función de asociación elegida podría producir una distribución no uniforme de las claves de búsqueda.

Para que la probabilidad de desbordamiento de cajones se reduzca, se escoge el número de cajones igual a $(n_r / f_r)*(1 + d)$, donde d es un factor normalmente con un valor cercano a 0,2. Se pierde algo de espacio: alrededor del 20 por ciento del espacio en los cajones estará vacío. Pero el beneficio es que la probabilidad de desbordamiento se reduce.

A pesar de la asignación de unos pocos más de cajones de los requeridos, el desbordamiento de cajones todavía puede suceder. Trataremos el desbordamiento de cajo-

nes utilizando **cajones de desbordamiento**. Si un registro se tiene que insertar en un cajón *c* y *c* está ya lleno, se proporcionará un cajón de desbordamiento para *c* y el registro se insertará en el cajón de desbordamiento. Si el cajón de desbordamiento está también lleno, se proporcionará otro cajón de desbordamiento, y así sucesivamente. Todos los cajones de desbordamiento de un cajón determinado están encadenados juntos en una lista enlazada, como se muestra en la Figura 12.22. El tratamiento del desbordamiento utilizando una lista enlazada se denomina **cadena de desbordamiento**.

Se debe variar un poco el algoritmo de búsqueda para tratar la cadena de desbordamiento. Como se dijo antes, la función de asociación se emplea sobre la clave de búsqueda para identificar un cajón *c*. Luego se examinan todos los registros en el cajón *c* para ver si coinciden con la clave búsqueda, como antes. Además, si el cajón *c* tiene cajones de desbordamiento, los registros en todos los cajones de desbordamiento de *c* se tienen que examinar también.

La forma de la estructura asociativa que se acaba de describir se denomina algunas veces **asociación cerrada**. Bajo una aproximación alternativa, llamada **asociación abierta**, se fija el conjunto de cajones y no hay cadenas de desbordamiento. Por el contrario, si un cajón está lleno, los registros se insertan en algún otro cajón del conjunto inicial *C* de cajones. Una política es utilizar el siguiente cajón (en orden cíclico) que tenga espacio; esta política se llama *ensayo lineal*. Se utilizan también otras políticas, tales como el cálculo funciones de asociación adicionales. La asociación abierta se emplea en la construcción de tablas de símbolos para compiladores y ensambladores, aunque es preferible la asociación cerrada para los sistemas de bases de datos. La razón es que el borrado bajo la asociación abierta es costoso. Normalmente, los compiladores y ensambladores realizan solamente operaciones de búsqueda e inserción en sus tablas de símbolos. Sin embargo, en un sistema de bases de datos es importante ser capaz de tratar el borrado tan bien como la inserción. Así, la asociatividad abierta es un aspecto de menor importancia en la implementación de bases de datos.

Un inconveniente importante relativo a la forma de asociación que se ha descrito es que la función de asociación se debe elegir cuando se implementa el sistema y no se puede cambiar fácilmente después si el archivo que se está indexando aumenta o disminuye. Ya que la función *h* asigna valores de la clave búsqueda a un conjunto fijo *C* de direcciones de cajón, emplearemos más espacio si *C* fue concebido para manejar el futuro crecimiento del archivo. Si *C* es demasiado pequeño, los cajones contienen registros de una gran variedad de valores de la clave búsqueda, pudiendo originar el desbordamiento del cajón. A medida que el archivo aumenta el rendimiento se degrada. Se estudiará más adelante, en el Apartado 12.6, cómo cambiar dinámicamente el número de cajones y la función de asociación.

12.5.2. Índices asociativos

La asociatividad se puede utilizar no solamente para la organización de archivos sino también para la creación de estructuras de índice. Un **índice asociativo** (*hash index*) organiza las claves de búsqueda, con sus punteros asociados, dentro de una estructura de archivo asociativo. Un índice asociativo se construye como se indica a continuación. Primero se aplica una función de asociación sobre la clave de búsqueda para identificar un cajón, luego se almacenan la clave y los punteros asociados en el cajón (o en los cajones de desbordamiento). En la Figura 12.23 se muestra un índice asociativo secundario en el archivo *cuenta* para la clave de búsqueda *número-cuen-*

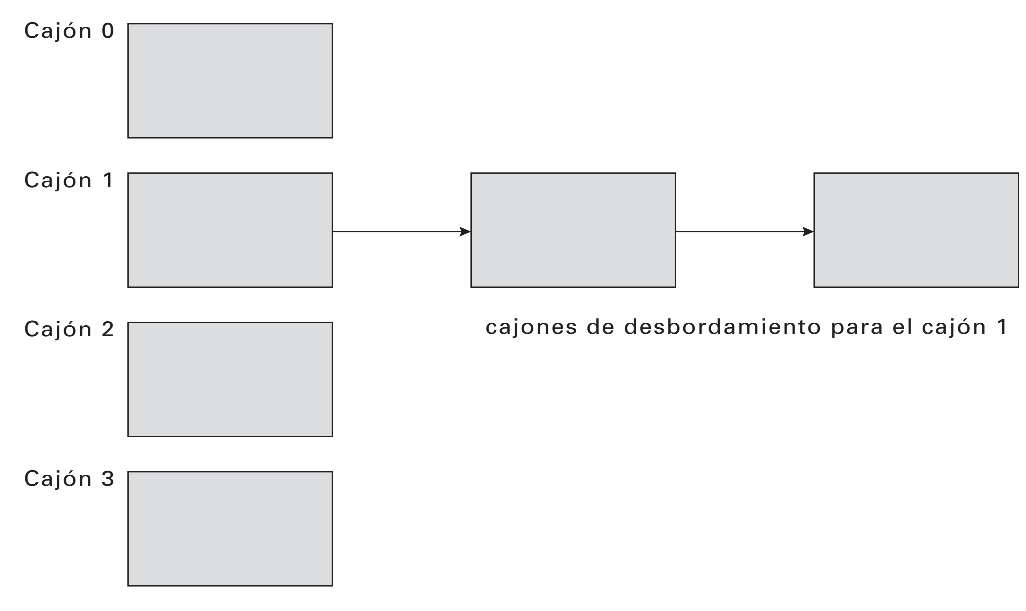


FIGURA 12.22. Cadena de desbordamiento en una estructura asociativa.

ta. La función de asociación utilizada calcula la suma de los dígitos del número de cuenta módulo siete. El índice asociativo tiene siete cajones, cada uno de tamaño dos (los índices realistas tendrían, por supuesto, tamaños de los cajones más grandes). Uno de los cajones tiene tres claves asignadas a él, por lo que tiene un cajón de desbordamiento. En este ejemplo, *número-cuenta* es una clave primaria para *cuenta*, así que cada clave de búsqueda tiene solamente un puntero asociado. En general se pueden asociar múltiples punteros con cada clave.

Se usa el término *índice asociativo* para denotar las estructuras de archivo asociativo, así como los índices secundarios asociativos. Estrictamente hablando, los índices asociativos son sólo estructuras de índices secundarios. Un índice asociativo nunca necesita una estructura de índice primario, ya que si un archivo está organizado utilizando asociatividad, no hay necesidad de una estructura de índice asociativo separada. Sin embargo, ya que la organización de archivos asociativos proporciona el mismo acceso directo a registros que se proporciona con la indexación, se pretende que la organización de un archivo mediante asociación también tenga un índice primario asociativo virtual en él.

12.6. ASOCIACIÓN DINÁMICA

Como se ha visto, la necesidad de fijar el conjunto *C* de direcciones de cajón presenta un problema serio con la técnica de asociación estática vista en el Apartado anterior. La mayoría de las bases de datos crecen con el tiempo. Si se va a utilizar la asociación estática para estas bases de datos, tenemos tres clases de opciones:

1. Elegir una función de asociación basada en el tamaño actual del archivo. Esta opción produci-

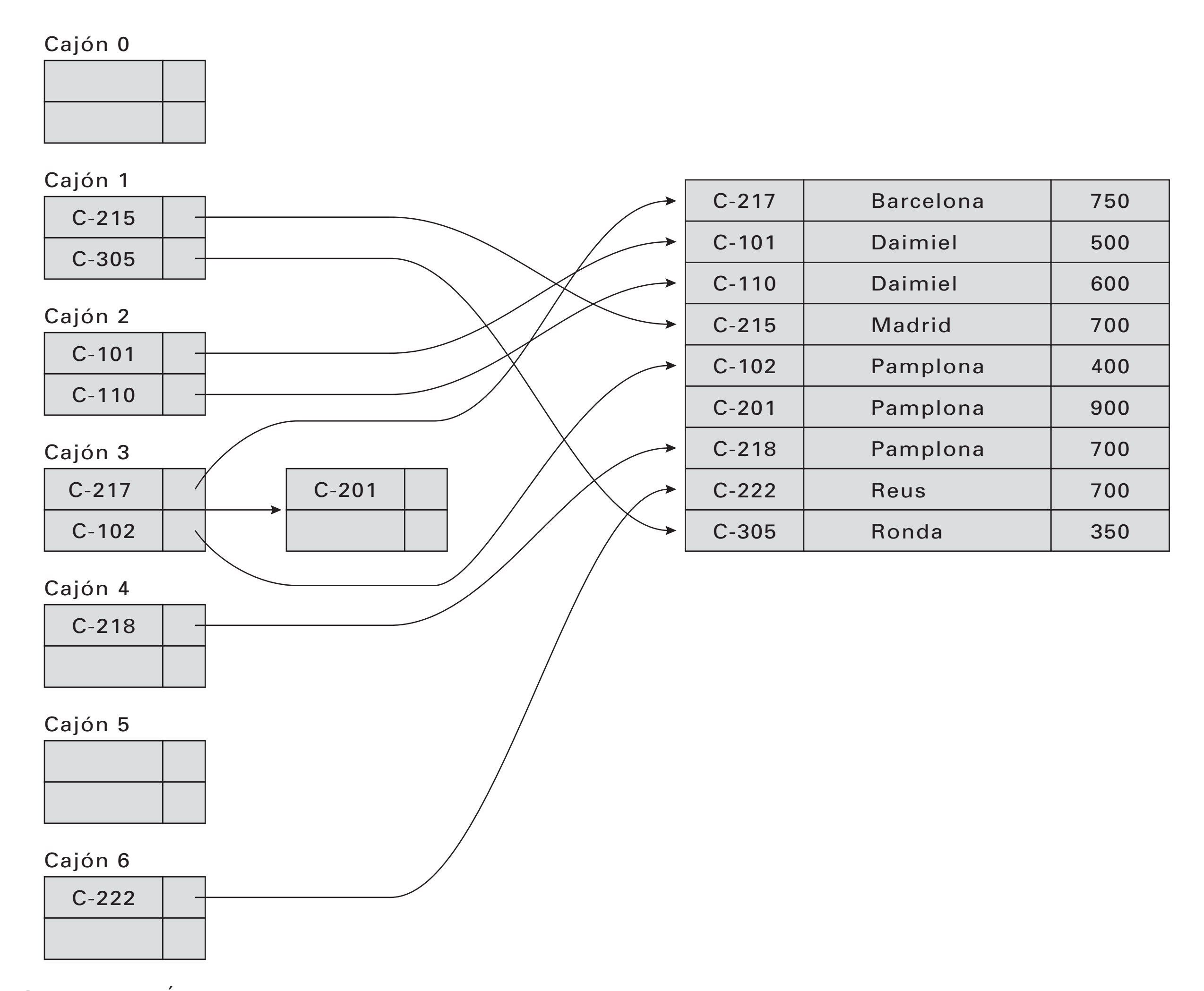


FIGURA 12.23. Índice asociativo de la clave de búsqueda *número-cuenta* del archivo *cuenta*.

rá una degradación del rendimiento a medida que la base de datos crezca.

2. Elegir una función de asociación basada en el tamaño previsto del archivo con relación a un punto determinado del futuro. Aunque se evite la degradación del rendimiento, inicialmente puede que se pierda una cantidad de espacio significante.
3. Reorganizar periódicamente la estructura asociativa en respuesta al crecimiento del archivo. Esta reorganización implica elegir una nueva función de asociación, volviendo a calcular la función de asociación de cada registro en el archivo y generando nuevas asignaciones de los cajones. Esta reorganización es una operación masiva que requiere mucho tiempo. Además, es necesario prohibir el acceso al archivo durante la reorganización.

Algunas técnicas de **asociación dinámica** permiten modificar la función de asociación dinámicamente para acomodarse al aumento o disminución de la base de datos. Describiremos una forma de asociación dinámica, llamada **asociación extensible**. Las notas bibliográficas proporcionan referencias a otras formas de asociatividad dinámica.

12.6.1. Estructura de datos

La asociación extensible hace frente a los cambios del tamaño de la base de datos dividiendo y fusionando los cajones a medida que la base de datos aumenta o disminuye. Como resultado se conserva eficazmente el espacio. Por otra parte, puesto que la reorganización se realiza sobre un cajón cada vez, la degradación del rendimiento resultante es aceptablemente baja.

Con la asociación extensible se elige una función de asociación h con las propiedades deseadas de uniformidad y aleatoriedad. Sin embargo, esta función de asociación genera valores dentro de un rango relativamente amplio, llamado, enteros binarios de b bits. Un valor normal de b es 32.

No se crea un cajón para cada valor de la función de asociación. De hecho, 2^{32} está por encima de 4 billones y no sería razonable crear tantos cajones salvo para las mayores bases de datos. Por el contrario, se crean tantos cajones bajo demanda, esto es, tantos como registros haya insertados en el archivo. Inicialmente no se utiliza el total de b bits del valor de la función de asociación. En cualquier caso, empleamos i bits, donde $0 \leq i \leq b$. Estos i bits son utilizados como desplazamiento en una tabla adicional con las direcciones de los cajones. El valor de i aumenta o disminuye con el tamaño de la base de datos.

En la Figura 12.24 se muestra una estructura general de asociación extensible. La i que aparece en la figura encima de la tabla de direcciones de los cajones indica que se requieren i bits de la función de asociación $h(K)$ para determinar el cajón apropiado para K. Obviamente, este número cambiará a medida que el archivo aumente. Aunque se requieren i bits para encontrar la entrada correcta en la tabla de direcciones de los cajones, algunas entradas consecutivas de la tabla podrían apuntar al mismo cajón. Todas estas entradas tendrán un prefijo común del valor de la función de asociación, aunque la longitud de este prefijo podría ser menor que i. Por lo tanto, se asocia con cada cajón un número entero que proporciona la longitud del prefijo común del valor de la función de asociación. En la Figura 12.24, el entero asociado con el cajón j aparece como i_j. El

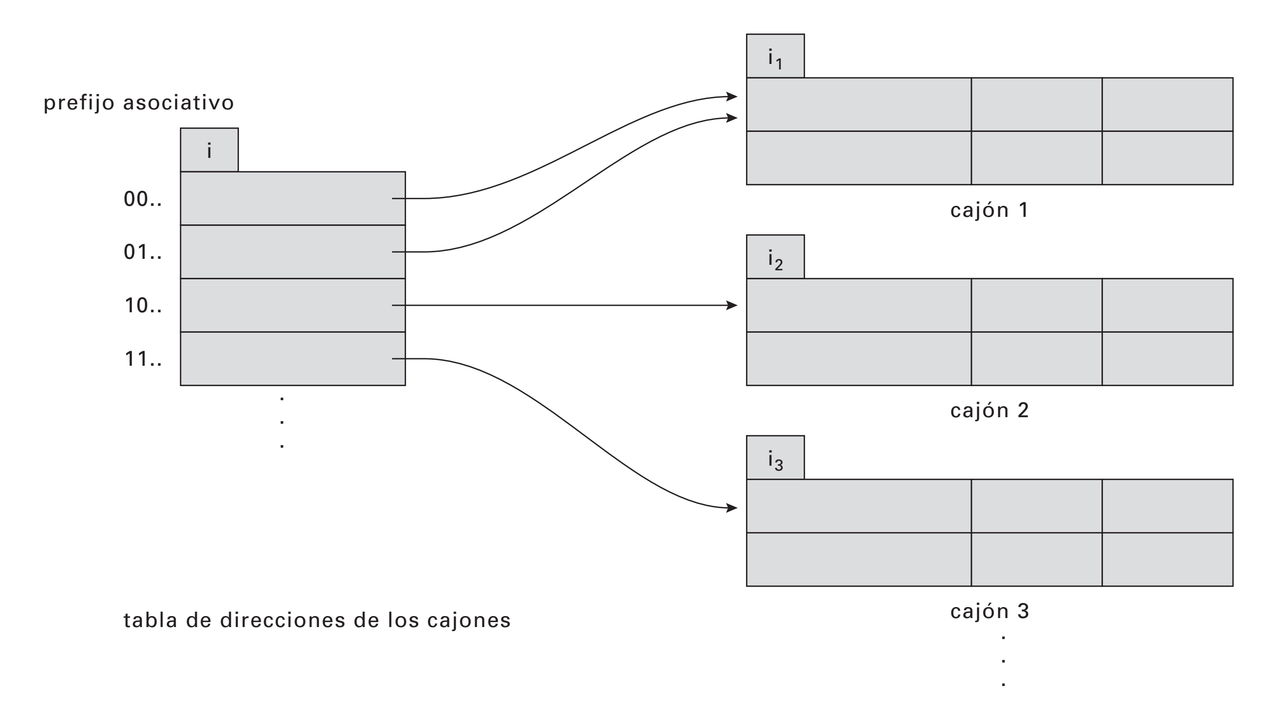


FIGURA 12.24. Estructura asociativa general extensible.

número de entradas en la tabla de direcciones de cajones que apuntan al cajón j es:

$$2^{(i - i_j)}$$

12.6.2. Consultas y actualizaciones

Ahora se verá cómo realizar la búsqueda, la inserción y el borrado en una estructura asociativa extensible.

Para localizar el cajón que contiene el valor de la clave de búsqueda K_l, se toman los primeros i bits más significativos de $h(K_l)$, se busca la entrada de la tabla que corresponda a esta cadena de bits, y se sigue el puntero del cajón en la entrada de la tabla.

Para insertar un registro con un valor de la clave de búsqueda K_l se sigue el mismo procedimiento de búsqueda que antes, llegando a algún cajón j. Si hay sitio en el cajón se inserta el registro en el cajón. Si por el contrario el cajón está lleno, hay que dividir el cajón y redistribuir los registros actuales más uno nuevo. Para dividir el cajón, primero hay que determinar del valor de la función de asociación por si fuera necesario incrementar el número de bits que se están usando.

- Si $i = i_j$, entonces solamente una entrada en la tabla de direcciones de los cajones apunta al cajón j. Por tanto, es necesario incrementar el tamaño de la tabla de direcciones de los cajones para incluir los punteros a los dos cajones que resultan de la división del cajón j. Esto se hace considerando un bit adicional en el valor de la función de asociación. Luego se incrementa el valor de i en uno, duplicando el tamaño de la tabla de direcciones de cajones. Cada entrada se sustituye por dos entradas, cada una de las cuales con el mismo puntero que la entrada original. Ahora dos entradas en la tabla de direcciones de cajones apuntan al cajón j. Así pues, se asigna un nuevo cajón (cajón z) y hacemos que la segunda entrada apunte al nuevo cajón. Se pone i_j e i_z a i. A continuación se vuelve a calcular la función de asociación para cada registro del cajón j y, dependiendo de los primeros i bits (recuérdese que se ha añadido uno a i), se mantiene en el cajón j o se coloca en el cajón recién creado.

  Ahora se vuelve a intentar la inserción del nuevo registro. Normalmente el intento tiene éxito. Sin embargo, si todos los registros del cajón j, así como el nuevo registro, tienen el mismo prefijo del valor de la función de asociación, será necesario dividir el cajón de nuevo, ya que todos los registros en el cajón j y el nuevo registro tienen asignados el mismo cajón. Si la función de asociación se eligió cuidadosamente, es poco probable que una simple inserción provoque que un cajón se divida más de una vez, a menos que haya un gran número de registros con la misma clave de búsqueda. Si todos los registros en el cajón j tienen el mismo valor de la clave de búsqueda, ningún número de divisiones servirá. En estos casos se usan cajones de desbordamiento para almacenar los registros, como en la asociación estática.

- Si $i > i_j$, entonces más de una entrada en la tabla de direcciones de cajones apunta al cajón j. Así, se puede dividir el cajón j sin incrementar el tamaño de la tabla de direcciones. Obsérvese que todas las entradas que apuntan al cajón j corresponden a prefijos del valor de la función de asociación que tienen el mismo valor en los i_j bits más a la izquierda. Se asigna un nuevo cajón (cajón z) y se pone i_j e i_z al valor que resulta de añadir uno al valor i_j original. A continuación, es necesario ajustar las entradas en la tabla de direcciones de cajones que anteriormente apuntaban al cajón j. (Nótese que con el nuevo valor de i_j no todas las entradas corresponden a prefijos del valor de la función de asociación que tienen el mismo valor en los i_j bits más a la izquierda). La primera mitad de todas las entradas se dejan como estaban (apuntando al cajón j) y el resto de entradas se ponen apuntando al cajón recién creado (cajón z). Por último, como en el caso anterior, se vuelve a calcular la función de asociación para cada registro en el cajón j y se colocan o bien en el cajón j o bien en el cajón z recién creado.

  Luego se vuelve a intentar la inserción. En el caso poco probable de que vuelva a fallar, se aplica uno de los dos casos, $i = i_j$ o $i > i_j$, según sea lo apropiado.

Nótese que en ambos casos sólo se necesita recalcular la función de asociación en los registros del cajón j.

Para borrar un registro con valor de la clave de búsqueda K_l se sigue el mismo procedimiento de búsqueda anterior, finalizando en algún cajón, llamémosle j. Se borran ambos, el registro del archivo y la clave de búsqueda del cajón. El cajón también se elimina si se queda vacío. Nótese que en este momento, varios cajones se pueden fusionar, reduciendo el tamaño de la tabla de direcciones de cajones a la mitad. El procedimiento para decidir cuándo y cómo fusionar cajones se deja como un ejercicio a realizar. Las condiciones bajo la que la tabla de direcciones de cajones se puede reducir de tamaño también se dejan como ejercicio. A diferencia de la fusión de cajones, el cambio de tamaño de la tabla de direcciones de cajones es una operación muy costosa si la tabla es grande. Por tanto, sólo sería aconsejable reducir el tamaño de la tabla de direcciones de cajones si el número de cajones se reduce considerablemente.

El ejemplo del archivo *cuenta* en la Figura 12.25 ilustra la operación de inserción. Los valores de la función de asociación de 32 bits para *nombre-sucursal* se mues-

C-217	Barcelona	750
C-101	Daimiel	500
C-110	Daimiel	600
C-215	Madrid	700
C-102	Pamplona	400
C-201	Pamplona	900
C-218	Pamplona	700
C-222	Reus	700
C-305	Ronda	350

FIGURA 12.25. Archivo de ejemplo *cuenta*.

tran en la Figura 12.26. Se asume que inicialmente el archivo está vacío, como se muestra en la Figura 12.27. Insertaremos los registros de uno en uno. Para mostrar todas las características de la asociación extensible se empleará una estructura pequeña y se hará la suposición no realista de que un cajón sólo puede contener dos registros.

Vamos a insertar el registro (C-217, Barcelona, 750). La tabla de direcciones de cajones contiene un puntero al único cajón donde se inserta el registro. A continuación, insertamos el registro (C-101, Daimiel, 500). Este registro también se inserta en el único cajón de la estructura.

Cuando se intenta insertar el siguiente registro (C-110, Daimiel, 600), nos encontramos con que el cajón está lleno. Ya que $i = i_0$, es necesario incrementar el número de bits que se toman del valor de la función de asociación. Ahora se utiliza un bit, permitiendo $2^1 = 2$ cajones. Este incremento en el número de bits necesarios duplica el tamaño de la tabla de direcciones de cajones a dos entradas. Se continúa con la división del cajón, colocando en el nuevo cajón aquellos registros cuya clave de búsqueda tiene un valor de la función de asociación que comienza por 1 y dejando el resto de registros en el cajón original. En la Figura 12.28 se muestra el estado de la estructura después de la división.

A continuación insertamos (C-215, Madrid, 700). Ya que el primer bit de *h*(Madrid) es 1, se tiene que insertar este registro en el cajón apuntado por la entrada «1» en la tabla de direcciones de cajones. Una vez más, nos encontramos con el cajón lleno e $i = i_1$. Se incrementa el número de bits que se usan del valor de la función de asociación a dos. Este incremento en el número de bits necesarios duplica el tamaño de la tabla de direcciones de cajones a cuatro entradas, como se muestra en la Figura 12.29. Como el cajón con el prefijo 0 del valor de la función de asociación de la Figura 12.28 no se dividió, las dos entradas de la tabla de direcciones 00 y 01 apuntan a este cajón.

Para cada registro en el cajón de la Figura 12.28 con prefijo 1 del valor de la función de asociación (el cajón que se dividió) se examinan los dos primeros bits del valor de la función de asociación para determinar qué cajón de la nueva estructura le corresponde.

Proseguimos insertando el registro (C-102, Pamplona, 400), el cual se aloja en el mismo cajón que Madrid. La siguiente inserción, la de (C-201, Pamplona, 900), provoca un desbordamiento en un cajón, produciendo el incremento del número de bits y la duplicación del tamaño de la tabla de direcciones de cajones. La inserción del tercer registro de Pamplona (C-218, Pamplona, 700) produce otro desbordamiento. Sin embargo, este desbordamiento no se puede resolver incrementando el número de bits, ya que los tres registros tienen

nombre-sucursal	*h(nombre-sucursal)*
Barcelona	0010 1101 1111 1011 0010 1100 0011 0000
Daimiel	1010 0011 1010 0000 1100 0110 1001 1111
Madrid	1100 0111 1110 1101 1011 1111 0011 1010
Pamplona	1111 0001 0010 0100 1001 0011 0110 1101
Reus	0011 0101 1010 0110 1100 1001 1110 1011
Ronda	1101 1000 0011 1111 1001 1100 0000 0001

FIGURA 12.26. Función de asociación para *nombre-sucursal*.

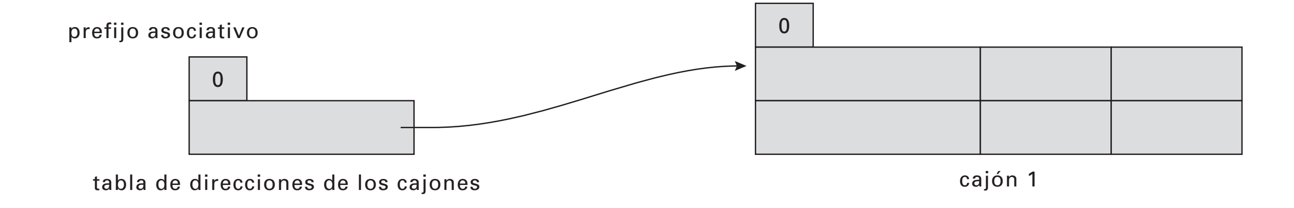


FIGURA 12.27. Estructura asociativa extensible inicial.

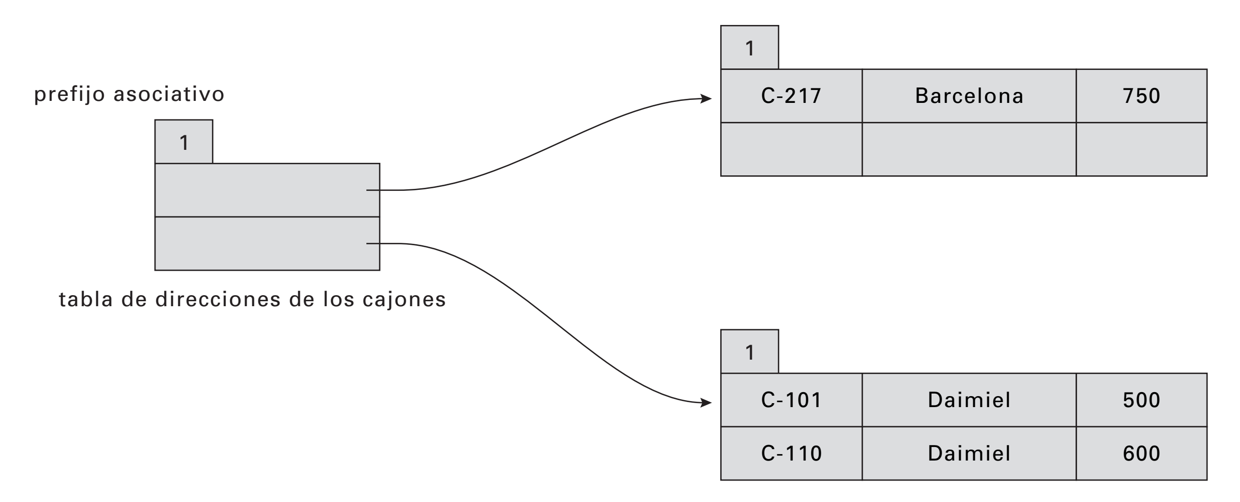


FIGURA 12.28. Estructura asociativa después de tres inserciones.

exactamente el mismo valor de la función de asociación. Por tanto, se utiliza un cajón de desbordamiento, como se muestra en la Figura 12.30.

Se continúa de esta manera hasta que se insertan todos los registros del archivo *cuenta* de la Figura 12.24. La estructura resultante se muestra en la Figura 12.31.

12.6.3. Comparaciones con otros esquemas

Examinemos a continuación las ventajas e inconvenientes de la asociación extensible frente a otros esquemas ya discutidos. La ventaja principal de la asociación extensible es que el rendimiento no se degrada según crece el archivo. Además de esto, el espacio adicional requerido es mínimo. Aunque la tabla de direcciones de cajones provoca un gasto adicional, sólo contiene un puntero por cada valor de la función de asociación con la longitud del prefijo actual. De esta manera el tamaño de la tabla es pequeño. El principal ahorro de espacio de la asociación extensible sobre otras formas de asociación es que no es necesario reservar cajones para un futuro crecimiento; en vez de ello se pueden asignar los cajones de manera dinámica.

Un inconveniente de la asociación extensible es que la búsqueda implica un nivel adicional de indirección,

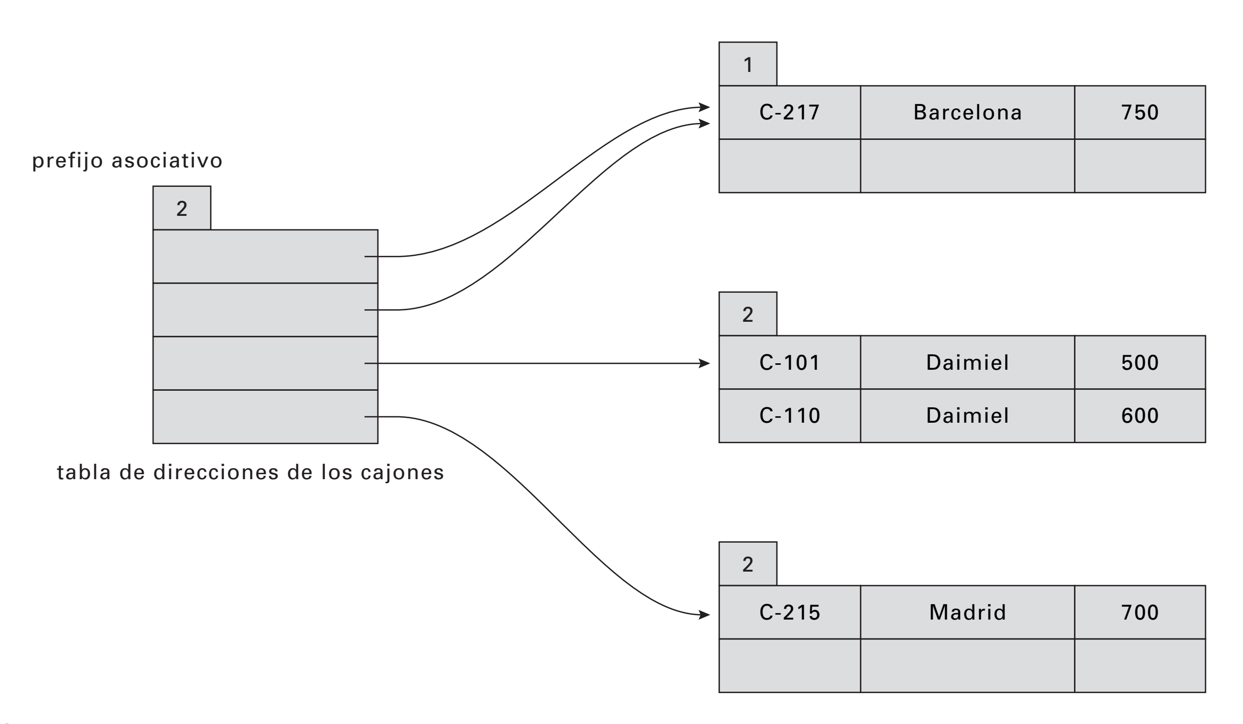


FIGURA 12.29. Estructura asociativa después de cuatro inserciones.

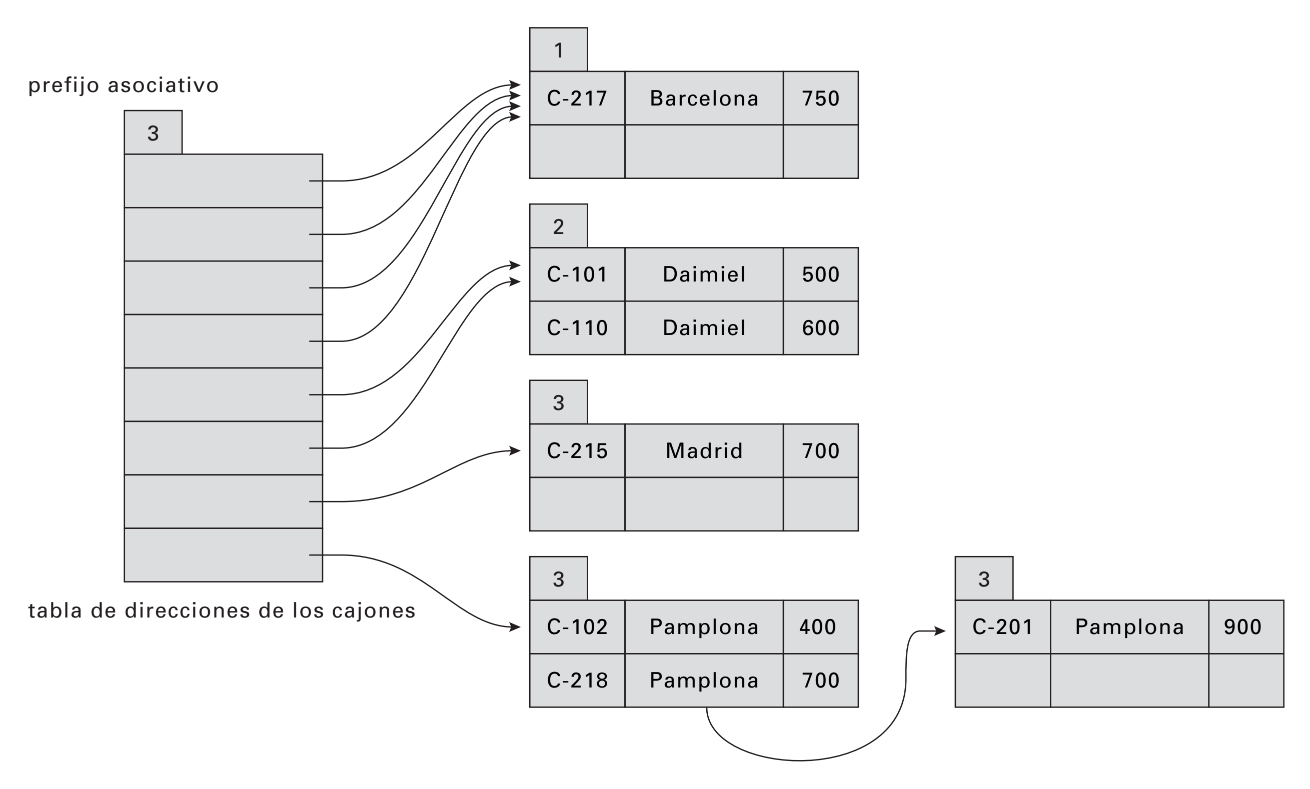


FIGURA 12.30. Estructura asociativa después de siete inserciones.

prefijo asociativo
3
tabla de direcciones de los cajones
1
C-217 Barcelona 750
C-222 Reus 700
2
C-101 Daimiel 500
C-110 Daimiel 600
3
C-215 Madrid 700
C-305 Ronda 350
3
C-102 Pamplona 400
C-218 Pamplona 700
3
C-201 Pamplona 900

FIGURA 12.31. Estructura asociativa extensible para el archivo *cuenta*.

ya que se debe acceder a la tabla de direcciones de los cajones antes que acceder al propio cajón. Esta referencia extra tiene una repercusión menor en el rendimiento. Aunque las estructuras asociativas que se discutieron con anterioridad no tienen este nivel extra de indirección, pierden su leve ventaja en el rendimiento cuando se llenan.

Por tanto, la asociación extensible se muestra como un técnica muy atractiva teniendo en cuenta que se acepta la complejidad añadida en su implementación. En las notas bibliográficas se proporcionan descripciones más detalladas sobre la implementación de la asociación extensible. Las notas bibliográficas también tienen referencias a otra forma de asociación dinámica llamada **asociación lineal**, la cual evita el nivel extra de indirección asociado con la asociación extensible y el posible coste de más cajones de desbordamiento.

12.7. COMPARACIÓN DE LA INDEXACIÓN ORDENADA Y LA ASOCIACIÓN

Se han visto varios esquemas de indexación ordenada y varios esquemas de asociación. Se pueden organizar los archivos de registros como archivos ordenados, utilizando una organización de índice secuencial o usando organizaciones de árbol B^+. Alternativamente se pueden organizar los archivos usando asociación. Finalmente, podemos organizar los archivos como montículos, donde los registros no están ordenados de ninguna manera en particular.

Cada esquema tiene sus ventajas dependiendo de la situación. Un implementador de un sistema de bases de datos podría proporcionar muchos esquemas, dejando al diseñador de la base de datos la decisión final de qué esquemas utilizar. Sin embargo, esta aproximación requiere que el implementador escriba más código, aumentando así el coste del sistema y el espacio que éste ocupa. Por tanto, la mayoría de los sistemas usan solamente unos pocos o sólo una forma de organización asociativa de archivos o índices asociativos.

Para hacer una sabia elección, el implementador o el diseñador de la base de datos debe considerar los siguientes aspectos:

- ¿Es aceptable el coste de una reorganización periódica del índice o de una estructura asociativa?
- ¿Cuál es la frecuencia relativa de las inserciones y borrados?
- ¿Es deseable mejorar el tiempo medio de acceso a expensas de incrementar el tiempo de acceso en el peor de los casos?
- ¿Qué tipos de consultas se supone que van a realizar los usuarios?

De estos puntos ya se han examinado los tres primeros, comenzando con la revisión de las ventajas relativas de las distintas técnicas de indexación y otra vez en la discusión de las técnicas de asociación. El cuarto punto, el tipo esperado de la consulta, es un aspecto crítico para la elección entre la indexación ordenada o la asociación.

Si la mayoría de las consultas son de la forma:

select $A_1, A_2, ..., A_n$
from r
where $A_i = c$

entonces, para procesar esta consulta, el sistema realizará una búsqueda en un índice ordenado o en una estructura asociativa de un atributo A_i con el valor c. Para este tipo de consultas es preferible un esquema asociativo. Una búsqueda en un índice ordenado requiere un tiempo proporcional al logaritmo del número de valores para A_i en r. Sin embargo, en una estructura asociativa, el tiempo medio de una búsqueda es una constante que no depende del tamaño de la base de datos. La única ventaja de un índice sobre una estructura asociativa con este tipo de consulta es que el tiempo de una búsqueda en el peor de los casos es proporcional al logaritmo del número de valores para A_i en r. Por el contrario, si se utiliza una estructura asociativa, el tiempo de una búsqueda en el peor de los casos es proporcional al número de valores para A_i en r. Sin embargo, al ser poco probable el peor caso de búsqueda (mayor tiempo de búsqueda) con la asociación, es preferible usar en este caso una estructura asociativa.

Las técnicas de índices ordenados son preferibles a las estructuras asociativas en los casos donde la consulta especifica un rango de valores. Estas consultas tienen el siguiente aspecto:

select $A_1, A_2, ..., A_n$
from r
where $A_i \leq c_2$ **and** $A_i \geq c_1$

En otras palabras, la consulta anterior encuentra todos los registros A_i con valores entre c_1 y c_2.

Consideremos cómo procesar esta consulta usando un índice ordenado. Primero se realiza una búsqueda en el valor c_1. Una vez que se ha encontrado un cajón que contiene el valor c_1, se sigue la cadena de punteros en el índice en orden para leer el siguiente cajón y continuamos de esta manera hasta encontrar c_2.

Si en vez de un índice ordenado se tiene una estructura asociativa, se puede realizar una búsqueda en c_1 y localizar el correspondiente cajón, pero en general no es fácil determinar el siguiente cajón que se tiene que examinar. La dificultad surge porque una buena función de asociación asigna valores aleatoriamente a los cajones. Por tanto, no existe la noción del «siguiente cajón en el orden». La razón por la que no se pueden encade-

nar cajones juntos según cierto orden en A_i es que cada cajón tiene asignado muchos valores de la clave de búsqueda. Ya que los valores están diseminados aleatoriamente según la función de asociación, es muy probable que el rango de valores esté esparcido a través de muchos cajones o tal vez en todos. Por esta razón se tienen que leer todos los cajones para encontrar las claves de búsqueda requeridas.

Normalmente se usa la indexación ordenada, a menos que se sepa de antemano que las consultas sobre un rango de valores van a ser poco frecuentes, en cuyo caso se utiliza la asociación. Las organizaciones asociativas son particularmente útiles para archivos temporales creados durante el procesamiento de consultas, siempre que se realicen búsquedas basadas en un valor de la clave, pero no consultas de rangos.

12.8. DEFINICIÓN DE ÍNDICES EN SQL

La norma SQL no proporciona al usuario o administrador de la base de datos ninguna manera de controlar qué índices se crean y se mantienen por el sistema de base de datos. Los índices no se necesitan para la corrección, ya que son estructuras de datos redundantes. Sin embargo, los índices son importantes para el procesamiento eficiente de las transacciones, incluyendo las transacciones de actualización y consulta. Los índices son también importantes para un cumplimiento eficiente de las ligaduras de integridad. Por ejemplo, las implementaciones típicas obligan a declarar una clave (Capítulo 6) mediante la creación de un índice con la clave declarada como la clave de búsqueda del índice.

En principio, un sistema de base de datos puede decidir automáticamente qué índices crear. Sin embargo, debido al coste en espacio de los índices, así como el efecto de los índices en el procesamiento de actualizaciones, no es fácil hacer una elección apropiada automáticamente sobre qué índices mantener. Por este motivo, la mayoría de las implementaciones de SQL proporcionan al programador control sobre la creación y eliminación de índices vía órdenes del lenguaje de definición de datos.

A continuación se ilustrará las sintaxis de estas órdenes. Aunque la sintaxis que se muestra se usa ampliamente y está soportada en muchos sistemas de bases de datos, no es parte de la norma SQL:1999. Las normas SQL (hasta SQL:1999, al menos) no dan soporte al control del esquema físico de la base de datos y aquí nos limitaremos al esquema lógico de la base de datos.

Un índice se crea mediante la orden **create index**, la cual tiene la forma

create index <nombre-índice> **on** <nombre-relación> (<lista-atributos>)

lista-atributos es la lista de atributos de la relación que constituye la clave de búsqueda del índice.

Para definir un índice llamado *índice-s* de la relación *sucursal* con la clave de búsqueda *nombre-sucursal*, se escribe

create index *índice-s* **on** *sucursal* (*nombre-sucursal*)

Si deseamos declarar que la clave de búsqueda es una clave candidata, hay que añadir el atributo **unique** a la definición del índice. Con esto, la orden

create unique index *índice-s* **on** *sucursal* (*nombre-sucursal*)

declara *nombre-sucursal* como una clave candidata de *sucursal*. Si cuando se introduce la orden **create unique index**, *nombre-sucursal* no es una clave candidata, se mostrará un mensaje de error y el intento de crear un índice fallará. Por otro lado, si el intento de crear el índice ha tenido éxito, cualquier intento de insertar una tupla que viole la declaración de clave fallará. Hay que observar que el carácter **unique** es redundante si la declaración **unique** de SQL estándar se soporta en el sistema de base de datos.

El nombre de índice especificado con el índice se necesita para hacer posible la eliminación (*drop*) de índices. La orden **drop index** tiene la forma

drop index <nombre-índice>

12.9. ACCESOS MULTICLAVE

Hasta ahora se ha asumido implícitamente que se utiliza solamente un índice (o tabla asociativa) para procesar una consulta en una relación. Sin embargo, para ciertos tipos de consultas es ventajoso el uso de múltiples índices si éstos existen.

12.9.1. Uso de varios índices de clave única

Asumamos que el archivo *cuenta* tiene dos índices: uno para el *nombre-sucursal* y otro para *saldo*. Consideremos la consulta : «Encontrar todos los números de cuen-

ta de la sucursal Pamplona con saldos igual a 1.000 €». Se escribe

> **select** *número-préstamo*
> **from** *cuenta*
> **where** *nombre-sucursal* = «Pamplona» **and**
> *saldo* = 1000

Hay tres estrategias posibles para procesar esta consulta:

1. Usar el índice en *nombre-sucursal* para encontrar todos los registros pertenecientes a la sucursal de Pamplona. Luego se examinan estos registros para ver si *saldo* = 1.000.
2. Usar el índice en *saldo* para encontrar todos los registros pertenecientes a cuentas con saldos de 1.000 €. Luego se examinan estos registros para ver si *nombre-sucursal* = «Pamplona.»
3. Usar el índice en *nombre-sucursal* para encontrar *punteros* a registros pertenecientes a la sucursal Pamplona. Y también usar el índice en *saldo* para encontrar los punteros a todos los registros pertenecientes a cuentas con un saldo de 1.000. Se realiza la intersección de estos dos conjuntos de punteros. Aquellos punteros que están en la intersección apuntan a los registros pertenecientes a la vez a Pamplona y a las cuentas con un saldo de 1.000 €.

La tercera estrategia es la única de las tres que aprovecha la ventaja de tener varios índices. Sin embargo, incluso esta estrategia podría ser una pobre elección si sucediera lo siguiente:

- Hay muchos registros pertenecientes a la sucursal Pamplona.
- Hay muchos registros pertenecientes a cuentas con un saldo de 1.000 €.
- Hay solamente unos cuantos registros pertenecientes a *ambos,* a la sucursal Pamplona y a las cuentas con un saldo de 1.000 €.

Si estas condiciones ocurrieran, se tendrían que examinar un gran número de punteros para producir un resultado pequeño. La estructura de índices denominada «índice de mapas de bits» acelera significativamente la operación de inserción usada en la tercera estrategia. Los índices de mapas de bits se describen en el Apartado 12.9.4.

12.9.2. Índices sobre varias claves

Una estrategia más eficiente para este caso es crear y utilizar un índice con una clave de búsqueda (*nombre-sucursal*, *saldo*), esto es, la clave de búsqueda consistente en el nombre de la sucursal concatenado con el saldo de la cuenta. La estructura del índice es la misma que para cualquier otro índice, con la única diferencia de que la clave de búsqueda no es un simple atributo sino una lista de atributos. La clave de búsqueda se puede representar como una tupla de valores, de la forma $(a_1, ..., a_n)$, donde los atributos indexados son $A_1, ..., A_n$. El orden de los valores de la clave de búsqueda es el *orden lexicográfico*. Por ejemplo, para el caso de dos atributos en la clave de búsqueda, $(a_1, a_2) < (a_1, a_2)$ si $a_1 < a_2$, o bien $a_1 = b_1$ y $a_2 < b_2$. El orden lexicográfico es básicamente el mismo orden alfabético de las palabras.

El empleo de una estructura de índice ordenado con múltiples atributos tiene algunas deficiencias. Como ilustración considérese la consulta

> **select** *número-préstamo*
> **from** *cuenta*
> **where** *nombre-sucursal* < «Pamplona» **and**
> *saldo* = 1000

Se puede responder a esta consulta usando un índice ordenado con la clave de búsqueda (*nombre-sucursal*, *saldo*) de la manera siguiente: para cada valor de *nombre-sucursal* que es menor que «Pamplona» alfabéticamente localizar los registros con un *saldo* de 1.000. Sin embargo, debido a la ordenación de los registros en el archivo, es probable que cada registro esté en un bloque diferente de disco, causando muchas operaciones de E/S.

La diferencia entre esta consulta y la anterior es que la condición en *nombre-sucursal* es una condición de comparación, en vez de una condición de igualdad.

Para acelerar el procesamiento en general de consultas con varias claves de búsqueda (las cuales pueden implicar una o más operaciones de comparación) se pueden emplear varias estructuras especiales. Se considerará la estructura de *archivos en retícula* en el Apartado 12.9.3. Hay otra estructura, denominada *árbol R*, que también se puede usar para este propósito. El árbol R es una extensión que se usa fundamentalmente con tipos de datos geográficos y se pospone su descripción hasta el Capítulo 23.

12.9.3. Archivos en retícula

En la Figura 12.32 se muestra una parte de un **archivo en retícula** para las claves de búsqueda *nombre-sucursal* y *saldo* en el archivo *cuenta*. El *array* bidimensional de la figura se llama *array en retícula* y los *arrays* unidimensionales se llaman *escalas lineales*. El archivo en retícula tiene un único *array* en retícula y una escala lineal para cada atributo de la clave de búsqueda.

Las claves de búsqueda se asignan a las celdas como se describe a continuación. Cada celda en el *array* en retícula tiene un puntero a un cajón que contiene valores de las claves de búsqueda y punteros a los registros. Sólo se muestran en la figura algunos de los cajones y punteros desde las celdas. Para conservar espacio se permite que varios elementos del *array* puedan apuntar al

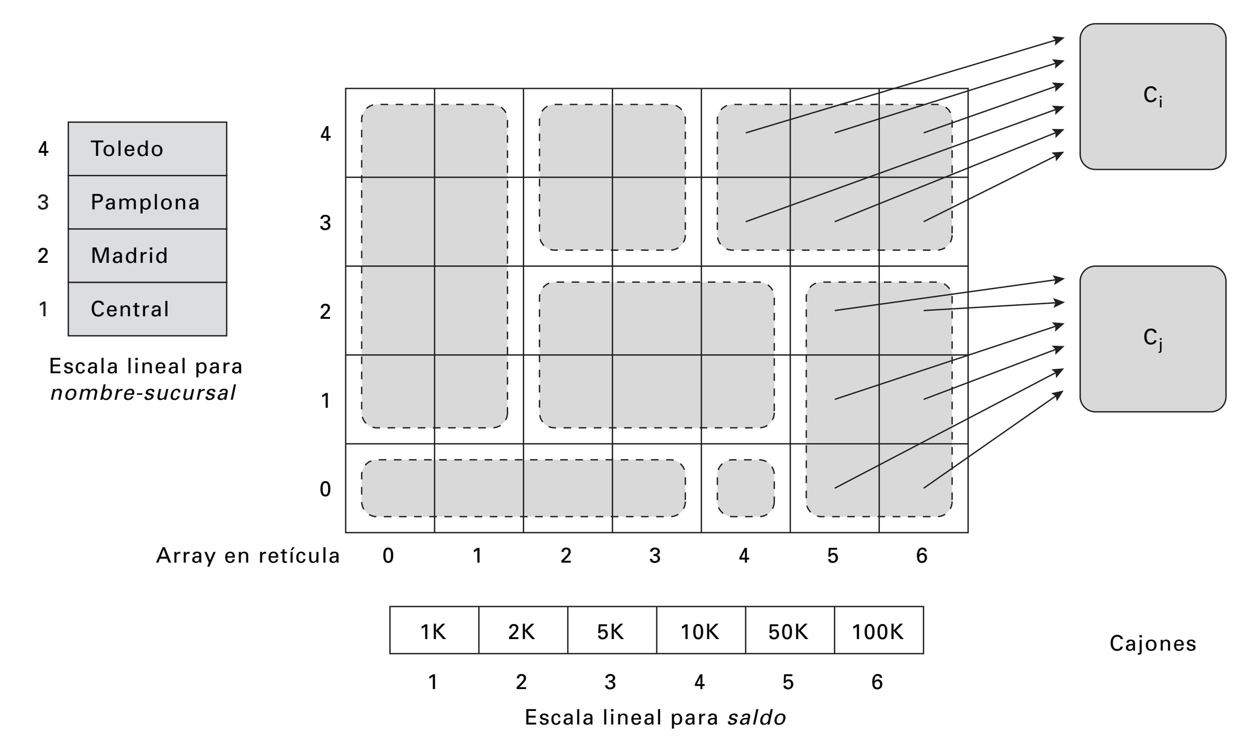


FIGURA 12.32. Archivo de rejilla para las claves *nombre-sucursal* y *saldo* del archivo *cuenta*.

mismo cajón. Los recuadros punteados de la figura señalan las celdas que apuntan al mismo cajón.

Supongamos que se quiere insertar en el índice de archivo en retícula un registro cuyo valor de la clave es («Barcelona», 500.000). Para encontrar la celda asignada a esta clave se localizan por separado la fila y la columna de la celda correspondiente.

Primero se utilizan las escalas lineales en *nombre-sucursal* para localizar la fila de la celda asignada a la clave de búsqueda. Para ello se busca en el *array* para encontrar el menor elemento que es mayor que «Barcelona». En este caso es el primer elemento, así que la fila asignada a la clave de búsqueda es la 0. Si fuera el *i*-ésimo elemento, la clave de búsqueda se asignaría a la fila *i*-1. Si la clave de búsqueda es mayor o igual que todos los elementos de la escala lineal, se le asignaría la última fila. A continuación se utiliza la escala lineal en *saldo* para encontrar de la misma manera qué columna le corresponde a la clave de búsqueda. En este caso, el saldo 500.000 tiene asignado la columna 6.

Por tanto, el valor de la clave de búsqueda («Barcelona», 500.000) tiene asignado la celda de la fila 0, columna 6. De la misma manera («Daimiel», 60.000) tendría asignada la celda de la fila 1, columna 5. Ambas celdas apuntan al mismo cajón (como se indica con el recuadro punteado), así que en los dos casos los valores de la clave de búsqueda y el puntero al registro están almacenados en el cajón con la etiqueta C_j de la figura.

Para realizar una búsqueda que responda a la consulta de nuestro ejemplo, con la condición de búsqueda

nombre-sucursal < «Pamplona» **and** *saldo* = 1000

buscamos todas las filas con nombres de sucursal menores que «Pamplona», utilizando la escala lineal de *nombre-sucursal*. En este caso, estas filas son la 0, 1 y 2. La fila 3 y posteriores contienen nombres de sucursal mayores o iguales a «Pamplona». De igual modo se obtiene que sólo la columna 1 puede tener un *saldo* de 1.000. En este caso sólo la columna 1 satisface esta condición. Así, solamente las celdas en la columna 1, filas 0, 1 y 2 pueden contener entradas que satisfagan la condición de búsqueda.

A continuación hay que examinar todas las entradas de los cajones apuntados por estas tres celdas. En este caso, solamente hay dos cajones, ya que dos de las celdas apuntan al mismo cajón, como se indica con los recuadros punteados de la figura. Los cajones podrían contener algunas claves de búsqueda que no satisfagan la condición requerida, de manera que cada clave de búsqueda del cajón se debe comprobar de nuevo para averiguar si satisface la condición o no. De cualquier modo, solamente hay que examinar un pequeño número de cajones para responder a la consulta.

Las escalas lineales se deben escoger de tal manera que los registros estén uniformemente distribuidos a través de las celdas. Si el cajón —llamémosle *A*— queda lleno y se tiene que insertar una entrada en él, se asigna un cajón adicional *B*. Si más de una celda apunta a *A*, se cambian los punteros a la celda de tal manera que algunos apunten a *A* y otros a *B*. Las entradas en el cajón *A* y la nueva entrada se redistribuyen entre *A* y *B* basán-

dose en las celdas que tengan asignados. Si sólo una celda apunta al cajón *A*, *B* se convierte en un cajón de desbordamiento de *A*. Para mejorar el rendimiento en esta situación se tiene que reorganizar el archivo en retícula con un *array* en retícula extendido y escalas lineales extendidas. Este proceso es como la expansión de la tabla de direcciones de los cajones en la asociación extensible y se deja a realizar como ejercicio.

Es conceptualmente sencillo extender la aproximación del archivo en retícula a cualquier número de claves de búsqueda. Si se quiere utilizar una estructura con *n* claves hay que construir un *array* en retícula *n*-dimensional con *n* escalas lineales.

La estructura de retícula es adecuada también para consultas que impliquen una sola clave de búsqueda. Considérese esta consulta:

> **select** *
> **from** *cuenta*
> **where** *nombre-sucursal* = «Pamplona»

La escala lineal de *nombre-sucursal* no dice que únicamente satisfacen esta condición las celdas de la fila 3. Como no hay condición según el *saldo*, se examinan todos los cajones apuntados por las celdas en la fila 3 para encontrar las entradas pertenecientes a Pamplona. De este modo, se puede usar un índice de archivo en retícula con dos claves de búsqueda para responder consultas en cada clave, así como para contestar consultas en ambas claves a la vez. Así un simple índice de archivo en retícula puede hacer el papel de tres índices distintos. Si cada índice se mantuviera por separado, los tres juntos ocuparían más espacio y el coste de su actualización sería mayor.

Los archivos en retícula proporcionan un descenso significante en el tiempo de procesamiento de consultas multiclave. Sin embargo, implican un gasto adicional de espacio (el directorio en retícula podría llegar a ser grande), así como una degradación del rendimiento al insertar y borrar registros. Además, es difícil elegir una división en los rangos de las claves para que la distribución de las claves sea uniforme. Si las inserciones en el archivo son frecuentes, la reorganización se tendrá que realizar periódicamente y eso puede tener un coste mayor.

12.9.4. Índices de mapas de bits

Los índices de mapas de bits son un tipo de índices especializado para la consulta sencilla sobre varias claves, aunqeu cada índice de mapas de bits se construya para una única clave.

Para que se usen los índices de mapas de bits, los registros de la relación deben estar numerados secuencialmente, comenzando, digamos, en 0. Dado un número *n* es fácil recuperar el registro con número *n*. Esto es particularmente fácil de conseguir si los registros tienen un tamaño fijo y están asignados en bloques consecutivos de un archivo. El número de registro se puede traducir fácilmente en un número de bloque y en un número que identifica el registro dentro del bloque.

Considérese una relación *r* con un atributo *A* que sólo puede valer un número pequeño (por ejemplo, entre 2 y 20). Por ejemplo, la relación *info-cliente* puede tener un campo *sexo*, que puede tomar sólo los valores m (masculino) o f (femenino). Otro ejemplo podría ser el atributo *nivel-ingresos*, donde los ingresos se han dividido en 5 niveles: *L*1: 0 – 9.999 €, *L*2: 10.000 – 19.999 €, *L*3: 20.000 – 39.999 €, *L*4: 40.000 – 74.999 y *L*5: 75.000 – ∞. Aquí, los datos originales pueden tomar muchos valores, pero un analista de datos debe dividir los valores en un número menor de rangos para simplificar el análisis de los datos.

12.9.4.1. *Estructura de los índices de mapas de bits*

Un **mapa de bits** es un *array* de bits. En su forma más simple, un **índice de mapas de bits** sobre un atributo *A* de la relación *r* consiste en un mapa de bits para cada valor que pueda tomar *A*. Cada mapa de bits tiene tantos bits como el número de registros de la relación. El *i*-ésimo bit del mapa de bits para el valor v_j se establece en 1 si el registro con número *i* tiene el valor v_j para el atributo *A*. El resto de bits del mapa de bits se establecen a 0.

En nuestro ejemplo, hay un mapa de bits para el valor m y otro para f. El *i*-ésimo bit del mapa de bits para m se establece en 1 si el valor *sexo* del registro con número *i* es m. El resto de bits del mapa de bits para m se establecen en 0. Análogamente, el mapa de bits para f tiene el valor 1 para los bits correspondientes a los registros con el valor f para el campo *sexo*; el resto de bits tienen el valor 0. La Figura 12.33 muestra un ejemplo de índices de mapa de bits para la relación *info-cliente*.

Ahora se considerará cuándo son útiles los mapas de bits. La forma más simple de recuperar todos los registros con el valor m (o el valor f) sería simplemente leer

número de registro	*nombre*	*sexo*	*dirección*	*nivel-ingresos*
0	Juan	m	Pamplona	L1
1	Diana	f	Barcelona	L2
2	María	f	Jaén	L1
3	Pedro	m	Barcelona	L4
4	Katzalin	f	Pamplona	L3

Mapas de bits para *sexo*

m	10010
f	01101

Mapas de bits para *nivel-ingresos*

L1	10100
L2	10100
L3	10100
L4	10100
L5	10100

FIGURA 12.33. Índices de mapas de bits para la relación *info-cliente*.

todos los registros de la relación y seleccionar los registros con el valor m (o f, respectivamente). El índice de mapas de bits no ayuda realmente a acelerar esta selección.

De hecho, los índices de mapas de bits son útiles para las selecciones principalmente cuando hay selecciones bajo varias claves. Supóngase que se crea un índice de mapas de bits sobre el atributo *nivel-ingresos*, que se describió antes, además del índice de mapas de bits para *sexo*.

Considérese ahora una consulta que seleccione mujeres con ingresos en el rango 10.000 – 19.999 €. Esta consulta se puede expresar como $\sigma_{sexo=\text{f} \wedge nivel\text{-}ingresos=L2}(\text{r})$. Para evaluar esta selección se busca el valor f en los mapas de bits de *sexo* y el valor *L*2 en los mapas de bits de *nivel-ingresos*, y se realiza la **intersección** (conjunción lógica) de los dos mapas de bits. En otras palabras, se calcula un nuevo mapa de bits donde el bit *i* tiene el valor 1 si el *i*-ésimo bit de los dos mapas de bits es 1, y tiene el valor 0 en caso contrario. En el ejemplo de la Figura 12.33, la intersección del mapa de bits para *sexo* = f (01101) y el mapa de bits para *nivel-ingresos* = *L*2 (01000) da como resultado el mapa de bits 01000.

Dado que el primer atributo puede tomar dos valores y el segundo cinco, se esperaría en media que entre 1 y 10 registros satisfarían la condición combinada de los dos atributos. Si hay más condiciones, la fracción de registros que satisfacen todas las condiciones será probablemente muy pequeña. El sistema puede calcular entonces el resultado de la consulta buscando todos los bits con valor 1 en el mapa de bits resultado de la intersección, y recuperando los registros correspondientes. Si la fracción es grande, la exploración de la relación completa seguiría siendo la alternativa menos costosa.

Otro uso importante de los mapas de bits es contar el número de tuplas que satisfacen una selección dada. Tales consultas son importantes para el análisis de datos. Por ejemplo, si deseamos determinar cuántas mujeres tienen un nivel de ingresos *L*2, se calcula la intersección de los dos mapas de bits y después se cuenta el número de bits que son 1 en la intersección. Así se puede obtener el resultado del mapa de bits sin acceder a la relación.

Los índices de mapas de bits son generalmente muy pequeños comparados con el tamaño real de la relación. Los registros tienen generalmente de decenas a centenas de bytes, mientras que un único bit representa a un registro en el mapa de bits. Así, el espacio ocupado por un único mapa de bits es usualmente menor que el 1 por ciento del espacio ocupado por la relación. Por ejemplo, si el tamaño de registro de una relación dada es 100 bytes, entonces el espacio ocupado por un único mapa de bits sería la octava parte del 1 por ciento del espacio ocupado por la relación. Si un atributo *A* de la relación puede tomar sólo uno de ocho valores, el índice de mapas de bits consistiría en 8 mapas de bits, que juntos ocupan sólo el 1 por ciento del tamaño de la relación.

El borrado de registros crea huecos en la secuencia de registros, ya que el desplazamiento de registros (o de los números de registro) para rellenar los huecos sería excesivamente costoso. Para reconocer los registros borrados se puede almacenar un **mapa de bits de existencia** en el que el bit *i* es 0 si el registro *i* no existe, y 1 en caso contrario. Se verá la necesidad de la existencia de los mapas de bits en el Apartado 12.9.4.2. La inserción de registros no afecta a la secuencia de numeración de otros registros. Por tanto, se puede insertar tanto añadiendo registros al final del archivo como reemplazando los registros borrados.

12.9.4.2. Implementación eficiente de las operaciones de mapas de bits

Se puede calcular fácilmente la intersección de dos mapas de bits usando un bucle **for**: la iteración *i*-ésima calcula la conjunción de los *i*-ésimos bits de los dos mapas de bits. Se puede acelerar considerablemente el cálculo de la intersección usando las instrucciones de bits **and** soportadas por la mayoría de los conjuntos de instrucciones de las computadoras. Una *palabra* consiste generalmente de 32 o 64 bits, dependiendo de la arquitectura de la computadora. Una instrucción de bits **and** toma dos palabras como entrada y devuelve una palabra en que cada bit es la conjunción lógica de los bits en las posiciones correspondientes de las palabras de entrada. Lo que es importante observar es que una única instrucción de bits **and** puede calcular la intersección de 32 o 64 bits *a la vez*.

Si una relación tuviese un millón de registros, cada mapa de bits contendría un millón de bits, o, equivalentemente, 128 Kbytes. Sólo se necesitan 31.250 instrucciones para calcular la intersección de dos mapas de bits para la relación, asumiendo un tamaño de palabra de 32 bits. Así, el cálculo de las intersecciones de mapas de bits es una operación extremadamente rápida.

Al igual que la intersección de mapas de bits es útil para calcular la conjunción de dos condiciones, la unión de mapas de bits es útil para calcular la disyunción de dos condiciones. El procedimiento para la unión de mapas de bits es exactamente igual que para la intersección, excepto en que se usa la instrucción de bits **or** en lugar de **and**.

La operación complemento se puede usar para calcular un predicado que incluya la negación de una condición, como **not**(*nivel-ingresos* = *L*1). El complemento de un mapa de bits se genera complementando cada bit del mapa de bits (el complemento de 1 es 0 y el complemento de 0 es 1). Puede parecer que **not**(*nivel-ingresos* = *L*1) se puede implementar simplemente calculando el complemento del mapa de bits para el nivel de ingresos *L*1. Sin embargo, si se han borrado algunos registros, el cálculo del complemento del mapa de bits no es suficiente. Los bits que correspondan a esos registros serían 0 en el mapa de bits original, pero serían 1 en el complemento, aunque el registro no exista. Tam-

bién aparece un problema similar cuando el valor de un atributo es *nulo*. Por ejemplo, si el valor de *nivel-ingresos* es nulo, el bit sería 0 en el mapa de bits original para el valor *L*1 y 1 en el complementado.

Para asegurarse de que los bits correspondientes a registros borrados se establezcan a 0 en el resultado, el mapa de bits complementado se debe intersectar con el mapa de bits de existencia para desactivar los bits de los registros borrados. Análogamente, para manejar los valores nulos, el mapa de bits complementado también se debe intersectar con el complemento del mapa de bits para el valor *nulo*[1].

El recuento del número de bits que son 1 en el mapa de bits se puede hacer fácilmente con una técnica inteligente. Se puede mantener un *array* de 256 entradas, donde la *i*-ésima entrada almacene el número de bits que son 1 en la representación binaria de *i*. Se establece el recuento inicial a 0. Se toma cada byte del mapa de bits, se usa para indexar en el *array* y se añade el recuento inicial al recuento total. El número de operaciones de suma sería la octava parte del número de tuplas, y así el proceso de recuento sería muy eficiente. Un gran *array* (que use 2^{16}=65.536 entradas), indexado por pares de bytes, daría incluso mayores aceleraciones pero a un mayor coste de almacenamiento.

12.9.4.3. Mapas de bits y árboles B$^+$

Los mapas de bits se pueden combinar con los índices normales de árboles B$^+$ para las relaciones donde unos pocos valores de atributo son extremadamente comunes, y otros valores también aparecen, pero con mucha menor frecuencia. En una hoja de un índice de un árbol B$^+$, para cada valor se mantendría normalmente una lista de todos los registros con ese valor para el atributo indexado. Cada elemento de la lista sería un identificador de registro, consistiendo en al menos 32 bits, y usualmente más. Para un valor que aparece en muchos registros, se almacena un mapa de bits en lugar de una lista de registros.

Supónganse que un valor particular v_i aparece en la dieciseisava parte de los registros de una relación, y también que los registros tienen un número de 64 bits que los identifica. El mapa de bits necesita sólo 1 bit por registro, o *N* en total. En cambio, la representación de lista necesita 64 bits por registro en el que aparezca el valor, o 64 * *N*/16 = 4*N* bits. Así, el mapa de bits es preferible para la representación de la lista de registros para el valor v_i. En el ejemplo (con un identificador de registro de 64 bits), si menos de 1 de cada 64 registros tiene un valor particular, es preferible la representación de lista de registros para la identificación de registros con ese valor, ya que usa menos bits que la representación con mapas de bits. Si más de 1 de cada 64 registros tiene un valor particular, la representación de mapas de bits es preferible.

Así, los mapas de bits se pueden usar como un mecanismo de almacenamiento comprimido en los nodos hoja de los árboles B$^+$, para los valores que aparecen muy frecuentemente.

12.10. RESUMEN

- Muchas consultas solamente hacen referencia a una pequeña proporción de los registros de un archivo. Para reducir el gasto adicional en la búsqueda de estos registros se pueden construir *índices* para los archivos almacenados en la base de datos.
- Los archivos secuenciales indexados son unos de los esquemas de índice más antiguos usados en los sistemas de bases de datos. Para permitir una rápida recuperación de los registros según el orden de la clave de búsqueda, los registros se almacenan consecutivamente y los que no siguen el orden se encadenan entre sí. Para permitir un acceso aleatorio, se emplean estructuras índice.
- Hay dos tipos de índices que se pueden utilizar: los índices densos y los índices dispersos. Los índices densos contienen una entrada por cada valor de la clave de búsqueda, mientras que los índices dispersos contienen entradas sólo para algunos de esos valores.
- Si el orden de una clave de búsqueda se corresponde con el orden secuencial del archivo, un índice sobre la clave de búsqueda se conoce como *índice primario*. Los otros índices son los *índices secundarios*. Los índices secundarios mejoran el rendimiento de las consultas que utilizan otras claves de búsqueda aparte de la primaria. Sin embargo, éstas implican un gasto adicional en la modificación de la base de datos.
- El inconveniente principal de la organización del archivo secuencial indexado es que el rendimiento disminuye según crece el archivo. Para superar esta deficiencia se puede usar un *índice de árbol B$^+$*.
- Un índice de árbol B$^+$ tiene la forma de un árbol *equilibrado*, en el cual cada camino de la raíz a las hojas del árbol tiene la misma longitud. La altura de un árbol

[1] La gestión de predicados tales como **is unknown** causaría aún más complicaciones, que requerirían en general el uso de un mapa de bits extra para determinar los resultados de las operaciones que son desconocidos.

B^+ es proporcional al logaritmo en base *N* del número de registros de la relación, donde cada nodo interno almacena *N* punteros; el valor de *N* está usualmente entre 50 y 100. Los árboles B^+ son más cortos que otras estructuras de árboles binarios equilibrados como los árboles AVL y, por tanto, necesitan menos accesos a disco para localizar los registros.

- Las búsquedas en un índice de árbol B^+ son directas y eficientes. Sin embargo, la inserción y el borrado son algo más complicados pero eficientes. El número de operaciones que se necesitan para la inserción y borrado en un árbol B^+ es proporcional al logaritmo en base *N* del número de registros de la relación, donde cada nodo interno almacena *N* punteros.
- Se pueden utilizar los árboles B^+ tanto para indexar un archivo con registros, como para organizar los registros de un archivo.
- Los índices de árbol B son similares a los índices de árbol B^+. La mayor ventaja de un árbol B es que el árbol B elimina el almacenamiento redundante de los valores de la clave de búsqueda. Los inconvenientes principales son la complejidad y el reducido grado de salida para un tamaño de nodo dado. En la práctica, los índices de árbol B^+ están casi generalmente mejor considerados que los índices de árbol B.
- Las organizaciones de archivos secuenciales necesitan una estructura de índice para localizar los datos. Los archivos con organizaciones basadas en asociación, en cambio, permiten encontrar la dirección de un elemento de datos directamente mediante el cálculo de una función con el valor de la clave de búsqueda del registro deseado. Ya que no se sabe en tiempo de diseño la manera precisa en la cual los valores de la clave de búsqueda se van a almacenar en el archivo, una buena función de asociación a elegir es la que distribuya los valores de la clave de búsqueda a los cajones de una manera uniforme y aleatoria.
- La *asociación estática* utiliza una función de asociación en la que el conjunto de direcciones de cajones está fijado. Estas funciones de asociación no se pueden adaptar fácilmente a las bases de datos que tengan un crecimiento significativo con el tiempo. Hay varias *técnicas de asociación dinámica* que permiten que la función de asociación cambie. Un ejemplo es la *asociación extensible,* que trata los cambios de tamaño de la base datos mediante la división y fusión de cajones según crezca o disminuya la base de datos.
- También se puede utilizar la asociación para crear índices secundarios; tales índices se llaman *índices asociativos*. Por motivos de notación se asume que las organizaciones de archivos asociativos tienen un índice asociativo implícito en la clave de búsqueda usada para la asociación.
- Los índices ordenados con árboles B^+ y con índices asociativos se pueden usar para la selección basada en condiciones de igualdad que involucren varios atributos. Cuando hay varios atributos en una condición de selección se pueden intersectar los identificadores de los registros recuperados con los diferentes índices.
- Los archivos en retícula proporcionan un medio general de indexación con múltiples atributos.
- Los índices de mapas de bits proporcionan una representación muy compacta para la indexación de atributos con muy pocos valores distintos. Las operaciones de intersección son extremadamente rápidas en los mapas de bits, haciéndolos ideales para el soporte de consultas con varios atributos.

TÉRMINOS DE REPASO

- Acceso con varias claves
- Árbol equilibrado
- Archivos en retícula
- Archivo secuencial indexado
- Asociación cerrada
- Asociación dinámica
- Asociación estática
- Asociación extensible
- Atasco
- Cajón
- Desbordamiento de cajones
- Espacio adicional
- Exploración secuencial
- Función de asociación
- Índice con agrupación
- Índice sin agrupación
- Índice de árbol B
- Índice de árbol B^+
- Índice asociativo
- Índice denso
- Índice disperso
- Índice de mapas de bits
- Índice multinivel
- Índice ordenado
- Índice primario
- Índice secundario

- Índices sobre varias claves
- Operaciones de mapas de bits
  - Intersección
  - Unión
  - Complemento
  - Mapa de bits de existencia
- Organización de archivos con árboles B^+
- Organización de archivos con asociación
- Registro/entrada del índice
- Tiempo de acceso
- Tiempo de borrado
- Tiempo de inserción
- Tipos de acceso

EJERCICIOS

12.1. ¿Cuándo es preferible utilizar un índice denso en vez de un índice disperso? Razónese la respuesta.

12.2. Dado que los índices agilizan el procesamiento de consultas, ¿por qué no deberían de mantenerse en varias claves de búsqueda? Enumérense tantas razones como sea posible.

12.3. ¿Cuál es la diferencia entre un índice primario y un índice secundario?

12.4. ¿Es posible en general tener dos índices primarios en la misma relación para dos claves de búsqueda diferentes? Razónese la respuesta.

12.5. Constrúyase un árbol B^+ con el siguiente conjunto de valores de la clave:

(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)

Asúmase que el árbol está inicialmente vacío y que se añaden los valores en orden ascendente. Constrúyanse árboles B^+ para los casos en los que el número de punteros que caben en un nodo son:

a. cuatro
b. seis
c. ocho

12.6. Para cada árbol B^+ del Ejercicio 12.5 muéstrense los pasos involucrados en las siguientes consultas:

a. Encontrar los registros con un valor de la clave de búsqueda de 11.
b. Encontrar los registros con un valor de la clave de búsqueda entre 7 y 17, ambos inclusive.

12.7. Para cada árbol B^+ del Ejercicio 12.5 muéstrese el aspecto del árbol después de cada una de las siguientes operaciones:

a. Insertar 9.
b. Insertar 10.
c. Insertar 8.
d. Borrar 23.
e. Borrar 19.

12.8. Considérese el esquema modificado de redistribución para árboles B^+ descrito en la página 295. ¿Cuál es la altura esperada del árbol en función de n?

12.9. Repítase el Ejercicio 12.5 para un árbol B.

12.10. Explíquense las diferencias entre la asociación abierta y la cerrada. Coméntense los beneficios de cada técnica en aplicaciones de bases de datos.

12.11. ¿Cuáles son las causas del desbordamiento de cajones en un archivo con una organización asociativa? ¿Qué se puede hacer para reducir la aparición del desbordamiento de cajones?

12.12. Supóngase que se está usando la asociación extensible en un archivo que contiene registros con los siguientes valores de la clave de búsqueda:

2, 3, 5, 7, 11, 17, 19, 23, 29, 31.

Muéstrese la estructura asociativa extensible para este archivo si la función de asociación es $h(x) = x \bmod 8$ y los cajones pueden contener hasta tres registros.

12.13. Muéstrese cómo cambia la estructura asociativa extensible del Ejercicio 12.12 como resultado de realizar los siguientes pasos:

a. Borrar 12.
b. Borrar 31.
c. Insertar 1.
d. Insertar 15.

12.14. Dese un pseudocódigo para el borrado de entradas de una estructura asociativa extensible, incluyendo detalles del momento y forma de fusionar cajones. No se debe considerar la reducción del tamaño de la tabla de direcciones de cajones.

12.15. Sugiérase una forma eficaz de comprobar si la tabla de direcciones de cajones en una asociación extensible se puede reducir en tamaño almacenando un recuento extra con la tabla de direcciones de cajones. Dense detalles de cómo se debería mantener el recuento cuando se dividen, fusionan o borran los cajones.

(Nota: la reducción del tamaño de la tabla de direcciones de cajones es una operación costosa y las inserciones subsecuentes pueden causar que la tabla vuelva a crecer. Por tanto, es mejor no reducir el tamaño tan pronto como se pueda, sino solamente si el número de entradas de índice es pequeño en comparación con el tamaño de la tabla de direcciones de cajones).

12.16. ¿Por qué una estructura asociativa no es la mejor elección para una clave de búsqueda en la que son frecuentes las consultas de rangos?

12.17. Considérese un archivo en retícula en el cual se desea evitar el desbordamiento de cajones por razones de rendimiento. En los casos en los que sería necesario un cajón de desbordamiento, en su lugar se reorganiza el archivo. Preséntese un algoritmo para esta reorganización.

12.18. Considérese la relación *cuenta* mostrada en la Figura 12.25.

a. Constrúyase un índice de mapa de bits sobre los atributos *nombre-sucursal* y *saldo*, dividiendo *saldo* en cuatro rangos: menores que 250, entre 250 y menor que 500, entre 500 y menor que 750, y 750 o mayor.

b. Considérese una consulta que solicite todas las cuentas de Daimiel con un saldo de 500 o más. Descríbanse los pasos para responder a la consulta y muéstrense los mapas de bits finales e intermedios construidos para responder la consulta.

12.19. Muéstrese la forma de calcular mapas de existencia a partir de otros mapas de bits. Asegúrese de que la técnica funciona incluso con valores nulos, usando un mapa de bits para el valor *nulo*.

12.20. ¿Cómo afecta el cifrado de datos a los esquemas de índices? En particular, ¿como afectaría a los esquemas que intentan almacenar los datos ordenados?

NOTAS BIBLIOGRÁFICAS

En Cormen et al. [1990] se pueden encontrar discusiones acerca de las estructuras básicas utilizadas en la indexación y asociación. Los índices de árbol B se introdujeron primero en Bayer [1972] y en Bayer y McCreight [1972]. Los árboles B^+ se discuten en Comer [1979], Bayer y Unterauer [1977] y Knuth [1973]. Las notas bibliográficas del Capítulo 16 proporcionan referencias a la investigación sobre los accesos concurrentes y las actualizaciones en los árboles B^+. Gray y Reuter [1993] proporcionan una buena descripción de los resultados en la implementación de árboles B^+.

Se han propuesto varias estructuras alternativas de árboles y basadas en árboles. Los *tries* son unos árboles cuya estructura está basada en los «dígitos» de las claves (por ejemplo, el índice de muescas de un diccionario, con una entrada para cada letra). Estos árboles podrían no estar equilibrados en el sentido que lo están los árboles B^+. Los *tries* se discuten en Ramesh et al. [1989], Orestein [1982], Litwin [1981] y Fredkin [1960]. Otros trabajos relacionados son los árboles B digitales de Lomet [1981].

Knuth [1973] analiza un gran número de técnicas de asociación distintas. Existen varias técnicas de asociación dinámica. Fagin et al. [1979] introduce la asociación extensible. La asociación lineal se introduce en Litwin [1978] y Litwin [1980]; en Larson [1982] se presentó un análisis del rendimiento de este esquema. Ellis [1987] examina la concurrencia con la asociación lineal. Larson [1988] presenta una variante de la asociación lineal. Otro esquema, llamado asociación dinámica se propone en Larson [1978]. Una alternativa propuesta en Ramakrishna y Larson [1989] permite la recuperación en un solo acceso a disco al precio de una gran sobrecarga en una pequeña fracción de las modificaciones de la base de datos. La asociación dividida es una extensión de la asociación para varios atributos, y se trata en Rivest [1976], Burkhard [1976] y Burkhard [1979].

La estructura de archivo en retícula aparece en Nievergelt et al [1984] y en Hinrichs [1985]. Los índices de mapas de bits y las variantes denominadas índices por capas de bits e índices de proyección se describen en O'Neil y Quass [1997]. Se introdujeron por primera vez en el gestor de archivos Model 204 de IBM sobre la plataforma AS 400. Proporcionan grandes ganancias de velocidad en ciertos tipos de consultas y se encuentran implementadas actualmente en la mayoría de sistemas de bases de datos. La investigación reciente en índices de mapas de bits incluye Wu y Buchmann [1998], Chan y Ioannidis [1998], Chan y Ioannidis [1999] y Johnson [1999a].

CAPÍTULO

13 PROCESAMIENTO DE CONSULTAS

El procesamiento de consultas hace referencia a la serie de actividades implicadas en la extracción de datos de una base de datos. Estas actividades incluyen la traducción de consultas expresadas en lenguajes de bases de datos de alto nivel en expresiones implementadas en el nivel físico del sistema, así como transformaciones de optimización de consultas y la evaluación real de las mismas.

13.1. VISIÓN GENERAL

En la Figura 13.1 se ilustran los pasos involucrados en el procesamiento de una consulta. Los pasos básicos son:

1. Análisis y traducción
2. Optimización
3. Evaluación

Antes de empezar el procesamiento de una consulta, el sistema debe traducir la consulta a una forma utilizable. Un lenguaje como SQL es adecuado para el uso humano, pero es poco apropiado para una representación interna en el sistema de la consulta. Así, una representación interna más útil está basada en el álgebra relacional extendida.

Así, la primera acción que el sistema tiene que emprender para procesar una consulta es la traducción de la consulta dada a su formato interno. Este proceso de traducción es similar al trabajo que realiza el analizador de un compilador. Durante la generación del formato interno de una consulta, el analizador comprueba la sintaxis de la consulta del usuario, verifica que los nombres de las relaciones que aparecen en la consulta sean nombres de relaciones en la base de datos, etcétera. Luego se construye un árbol para el análisis de la consulta, que se transformará en una expresión del álgebra relacional. Si la consulta estuviera expresada en términos de una vista, la fase de traducción también sustituye todos los usos de la vista mediante expresiones del álgebra relacional que definen la vista[1]. El análisis de lenguajes se describe en la mayoría de los libros sobre compiladores (véanse las notas bibliográficas).

Dada una consulta, hay generalmente varios métodos distintos para obtener la respuesta. Por ejemplo, ya se ha visto que en SQL se puede expresar una consulta de diferentes maneras. Cada consulta en SQL se puede traducir en una expresión del álgebra relacional de varias formas. Además de esto, la representación de una consulta en el álgebra relacional especifica de manera parcial cómo evaluar la consulta; hay normalmente varias maneras de evaluar expresiones del álgebra relacional. Como ejemplo, considérese la consulta

select *saldo*
from cuenta
where *saldo* < 2500

Esta consulta se puede traducir en alguna de las siguientes expresiones del álgebra relacional:

- $\sigma_{saldo < 2500} (\prod_{saldo} (cuenta))$
- $\prod_{saldo} (\sigma_{saldo < 2500} (cuenta))$

Además, se puede ejecutar cada operación del álgebra relacional utilizando alguno de los diferentes algoritmos. Por ejemplo, para implementar la selección anterior se puede examinar cada tupla en *cuenta* para encontrar las tuplas cuyo saldo sea menor que 2.500. Por otro lado, si se dispone de un índice de árbol B^+ en el atributo *saldo*, se puede utilizar este índice para localizar las tuplas.

Para especificar completamente cómo evaluar una consulta, no basta con proporcionar la expresión del álgebra relacional, además hay que anotar en ellas las instrucciones que especifiquen cómo evaluar cada ope-

[1] Para vistas materializadas, la expresión que define la vista ha sido ya evaluada y almacenada. Por tanto, se puede usar la relación almacenada en lugar de reemplazar los usos de la vista por la expresión que define la vista. Las vistas recursivas se tratan de manera diferente, mediante un procedimiento de búsqueda de punto fijo, según se vio en el Apartado 5.2.6.

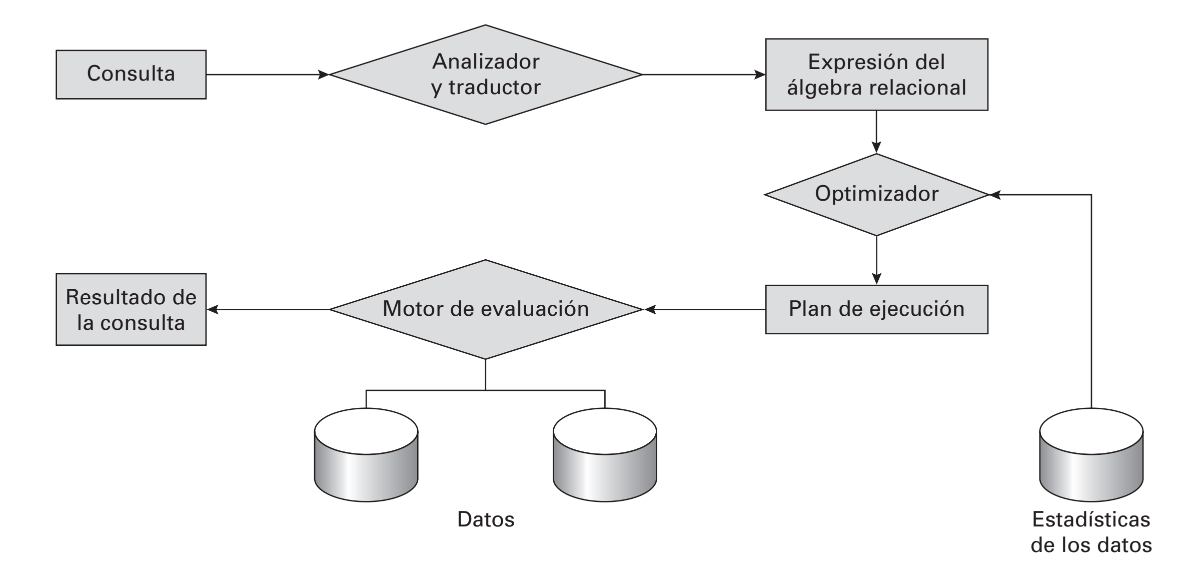


FIGURA 13.1. Pasos en el procesamiento de una consulta.

ración. Estas anotaciones podrían ser el algoritmo a usar para una operación específica o el índice o índices concretos a utilizar. Las operaciones del álgebra relacional anotadas con instrucciones sobre la evaluación reciben el nombre de **primitivas de evaluación**. Una secuencia de operaciones primitivas que se pueden utilizar para evaluar una consulta establecen un **plan de ejecución de la consulta** o **plan de evaluación de la consulta**. En la Figura 13.2 se ilustra un plan de evaluación para nuestro ejemplo de consulta, en el que se especifica un índice concreto (denotado en la figura como «índice 1») para la operación selección. El **motor de ejecución de consultas** toma un plan de evaluación, lo ejecuta y devuelve su respuesta a la consulta.

Los diferentes planes de evaluación para una consulta dada pueden tener costes distintos. No se puede esperar que los usuarios escriban las consultas de manera que sugieran el plan de evaluación más eficiente. En su lugar, es responsabilidad del sistema construir un plan de evaluación de la consulta que minimice el coste de la evaluación de la consulta. El Capítulo 14 describe en detalle la optimización de consultas.

Una vez que está elegido el plan de la consulta se evalúa la misma con ese plan y se muestra el resultado de la consulta.

La secuencia de pasos que se han descrito para procesar una consulta son representativos; no todas las bases de datos los siguen exactamente. Por ejemplo, en lugar de utilizar la representación del álgebra relacional, varias bases de datos usan una representación anotada del árbol de análisis basada en la estructura de la consulta SQL. Sin embargo, los conceptos que se describen aquí forman las bases del procesamiento de consultas en las bases de datos.

Para optimizar una consulta, el optimizador de consultas debe conocer el coste de cada operación. Aunque el coste exacto es difícil de calcular, dado que depende de muchos parámetros como la memoria real disponible, es posible obtener una estimación aproximada del coste de ejecución para cada operación.

En el Apartado 13.2 se describe cómo medir el coste de una consulta. Desde el Apartado 13.3 hasta el 13.6 se estudia la evaluación óptima de operaciones individuales. Varias operaciones se pueden agrupar en un **cauce**, en el que cada una de las operaciones empieza trabajando sobre sus tuplas de entrada del mismo modo que si fuesen generadas por otra operación. En el Apartado 13.7 se examina cómo coordinar la ejecución de varias operaciones de un plan de evaluación de consultas, en particular cómo usar las operaciones encauzadas para evitar escribir resultados intermedios en disco.

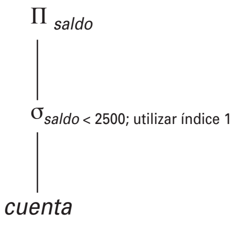


FIGURA 13.2. Plan de ejecución de una consulta.

13.2. MEDIDAS DEL COSTE DE UNA CONSULTA

El coste de la evaluación de una consulta se puede expresar en términos de diferentes recursos, incluyendo los accesos a disco, el tiempo de UCP en ejecutar una consulta y, en sistemas de bases de datos distribuidos o paralelos, el coste de la comunicación (que se discutirá más tarde en los Capítulos 19 y 20). El tiempo de respuesta para un plan de evaluación de una consulta (esto es, el tiempo de reloj que se necesita para ejecutar el plan), si se supone que no hay otra actividad ejecutándose en el sistema, podría tener en cuenta todos estos costes y utilizarlos como una buena medida del coste del plan.

Sin embargo, en grandes sistemas de bases de datos, los accesos a disco (que se miden como el número de transferencias de bloques de disco) son normalmente el coste más importante, ya que los accesos a disco son más lentos comparados con las operaciones en memoria. Además, la velocidad de la UCP está aumentando mucho más rápidamente que las velocidades de los discos. Así, lo más probable es que el tiempo empleado en operaciones del disco siga influenciando el tiempo total de ejecución de una consulta. Por último, las estimaciones del tiempo de UCP son más difíciles de hacer, comparadas con las estimaciones del coste de los accesos a disco. Por este motivo se considera el coste de los accesos a disco una medida razonable del coste del plan de evaluación de una consulta.

Se utilizará simplemente el término *número de transferencias de bloques* de disco como una medida del coste real. Para simplificar los cálculos del coste de los accesos a disco se asume que todas las transferencias de bloques tienen el mismo coste. Esta suposición ignora la varianza que surge de la latencia rotacional (esperando a que el dato deseado gire bajo la cabeza de lectura-escritura) y el tiempo de búsqueda (el tiempo que se emplea en mover la cabeza sobre el cilindro o pista deseada). Para obtener números más precisos es necesario distinguir entre **E/S secuencial**, donde los bloques leídos son contiguos en el disco, y la **E/S aleatoria**, donde los bloques no son contiguos, y es necesario pagar un coste extra de búsqueda por cada operación E/S. También es necesario distinguir entre las lecturas y escrituras de bloques, dado que se tarda más tiempo en escribir un bloque que leerlo de disco. Una medida más precisa sería estimar:

1. El número de operaciones de búsqueda realizadas.
2. El número de bloques leídos.
3. El número de bloques escritos.

y entonces sumar estos números después de multiplicarlos respectivamente por el tiempo medio de búsqueda, el tiempo medio de transferencia para la lectura de un bloque y el tiempo de transferencia para escribir un bloque. Los optimizadores de consultas de la vida real también tienen en cuenta los costes de UCP al calcular el coste de una operación. Por simplicidad se ignoran estos detalles y se deja como ejercicio para el lector obtener estimaciones del coste más precisas para las diferentes operaciones.

Las estimaciones de coste que se proporcionan ignoran el coste de escribir el resultado final de una operación en disco. Esto se tiene en cuenta aisladamente cuando sea preciso. Los costes de todos los algoritmos que se consideran aquí dependen del tamaño de la memoria intermedia en la memoria principal. En el mejor caso, todos los datos se pueden leer en las memorias intermedias y no es necesario acceder a disco de nuevo. En el peor caso se asume que la memoria intermedia puede contener sólo unos pocos bloques de datos, aproximadamente un bloque por relación. Al presentar las estimaciones de coste se asume generalmente el peor caso.

13.3. OPERACIÓN SELECCIÓN

En el procesamiento de consultas, el **explorador de archivo** es el operador de nivel más bajo para acceder a los datos. Los exploradores de archivo son algoritmos de búsqueda que localizan y recuperan los registros que cumplen una condición de selección. En los sistemas relacionales, el explorador de archivo permite leer una relación completa en esos casos donde la relación se almacena en un único archivo dedicado.

13.3.1. Algoritmos básicos

Considérese una operación selección en una relación cuyas tuplas se almacenan juntas en un archivo. A continuación se muestran dos algoritmos exploradores que implementan la operación selección:

- **A1 (búsqueda lineal).** En una búsqueda lineal se explora cada bloque del archivo y se comprueban todos los registros para ver si satisfacen la condición de selección. Para una selección sobre un atributo clave, el sistema puede terminar la exploración si se encuentra el registro requerido sin necesidad de examinar los otros registros de la relación.

  El coste de la búsqueda lineal, en términos de operaciones E/S es b_r, donde b_r denota el número

de bloques del archivo. Las selecciones sobre atributos clave tienen un coste medio de $b_r / 2$, pero aún tiene el coste en el peor caso de b_r.

Aunque podría ser más lento que otros algoritmos para la implementación de la selección, el algoritmo de búsqueda lineal se puede aplicar a cualquier archivo, sin importar la ordenación del archivo, la presencia de índices o la naturaleza de la operación selección. Los otros algoritmos que se estudiarán no son aplicables en todos los casos, pero cuando lo son, son más rápidos que la búsqueda lineal.

- **A2 (búsqueda binaria).** Si el archivo está ordenado según un atributo y la condición de la selección es una comparación de igualdad en ese atributo, se puede utilizar una búsqueda binaria para localizar los registros que satisfacen la selección. El sistema realiza la búsqueda binaria de los bloques del archivo.

  El número de bloques que deben ser examinados para encontrar los registros requeridos es $\lceil \log_2 (b_r) \rceil$, donde b_r denota el número de bloques del archivo. Si la selección no es sobre un atributo clave, más de un bloque puede contener los registros requeridos, y el coste de la lectura de los bloques extra se debe añadir a la estimación del coste. Se puede estimar este número mediante la estimación del tamaño del resultado de la selección (que se explica en el Apartado 14.2) y dividiéndolo entre el número de registros que se almacenan por bloque de la relación.

13.3.2. Selecciones con índices

Las estructuras índice se denominan **caminos de acceso**, ya que proporcionan un camino a través del cual se pueden localizar y acceder a los datos. En el Capítulo 12 se señaló la eficiencia de leer los registros del archivo en un orden próximo al orden físico. Recuérdese que un *índice primario* es un índice que permite leer los registros de un archivo en un orden que se corresponde con la ordenación física del archivo. Un índice que no es primario se llama *índice secundario*.

Los algoritmos de búsqueda que utilizan un índice reciben el nombre de **exploraciones del índice**. Los índices ordenados, como los árboles B^+, también permiten acceder a las tuplas según cierto orden que es útil para la implementación de las consultas de rangos. Aunque los índices pueden proporcionar un acceso rápido, directo y ordenado, su utilización implica un gasto adicional en los accesos a los bloques que contienen el índice. Utilizaremos el predicado de selección como guía en la selección del índice a usar en el procesamiento de la consulta. Los algoritmos de búsqueda que usan un índice son:

- **A3 (índice primario, igualdad basada en la clave).** Para una condición de igualdad en un atributo clave con un índice primario se puede utilizar el índice para recuperar el único registro que satisface la correspondiente condición de igualdad.

  Si se usa un árbol B^+, el coste de la operación en términos de operaciones E/S es igual a la altura del árbol más una operación E/S para recuperar el registro.
- **A4 (índice primario, igualdad basada en un atributo no clave).** Se pueden recuperar varios registros mediante el uso de un índice primario cuando la condición de selección especifica una comparación de igualdad en un atributo *A* que no sea clave. La única diferencia del caso anterior es que puede ser necesario recuperar varios registros. Sin embargo, estos registros estarían almacenados consecutivamente en el archivo, ya que el archivo se ordena según la clave de búsqueda.
- **A5 (índice secundario, igualdad).** Las selecciones con una condición de igualdad pueden utilizar un índice secundario. Esta estrategia puede recuperar un único registro si la condición de igualdad es sobre una clave; puede que se recuperen varios registros si el campo índice no es clave.

  En el primer caso sólo se obtiene un registro, y el coste es igual a la altura del árbol más una operación E/S para recuperar el registro. En el segundo caso, cada registro puede residir en un bloque diferente, que puede resultar en una operación E/S por cada registro recuperado. El coste podría llegar a ser incluso peor que el de la búsqueda lineal si se obtiene un gran número de registros.

  Supongamos que se utiliza la misma información estadística de la relación *cuenta* utilizada en el ejemplo anterior. Supóngase también la existencia de los siguientes índices en la relación cuenta.

  - Un índice árbol B^+ primario para el atributo *nombre-sucursal*
  - Un índice árbol B^+ secundario para el atributo *saldo* +

  Como se mencionó anteriormente, se asume la simplificación de que los valores se distribuyen de manera uniforme.

Si se usan organizaciones de archivos de árboles B^+ para almacenar relaciones, los registros se puede trasladar entre los bloques cuando los nodos hoja se dividen o combinan y cuando se redistribuyen. Si los índices secundarios almacenan punteros a la ubicación física de los registros, los punteros tendrán que actualizarse cuando se trasladen los registros. En algunos sistemas, como Non-Stop SQL System de Compaq, los índices secundarios almacenan el valor de la clave en la organización de archivos de árboles B^+. El acceso a un registro mediante un índice secundario es incluso más caro, dado que se debe realizar la búsqueda en el árbol B^+ usado en la organización de archivos. La fórmula del

coste descrita para índices secundarios tendrá que ser modificada adecuadamente si se usan tales índices.

13.3.3. Selecciones con condiciones de comparación

Considérese una selección de la forma $\sigma_{A \leq v}(r)$. Se pueden implementar utilizando búsqueda lineal o binaria, o con índices de alguna de las siguientes maneras:

- **A6 (índice primario, comparación).** Se puede utilizar un índice ordenado primario (por ejemplo, un índice primario de árbol B^+) cuando la condición de selección es una comparación. Para condiciones con comparaciones de la forma $A > v$ o $A \geq v$ se puede usar el índice primario para guiar la recuperación de las tuplas de la manera siguiente. Para el caso de $A \geq v$ se busca el valor de v en el índice para encontrar la primera tupla del archivo que tenga un valor de $A = v$. Un explorador de archivo comenzando en esa tupla y hasta el final del archivo devuelve todas las tuplas que satisfacen la condición. Para $A > v$, el explorador de archivo comienza con la primera tupla tal que $A > v$.

  Para comparaciones de la forma $A < v$ o $A \leq v$ no es necesario buscar en el índice. Para el caso de $A < v$, se utiliza un simple explorador de archivo partiendo del inicio del archivo y continuando hasta (pero sin incluirlo) la primera tupla con el atributo $A = v$. El caso $A \leq v$ es similar, excepto que el explorador continúa hasta (pero sin incluir) la primera tupla con el atributo $A > v$. Para ambos casos el índice no es de utilidad alguna.
- **A7 (índice secundario, comparación).** Se puede utilizar un índice secundario ordenado para guiar la recuperación bajo condiciones de comparación que contengan $<$, $\leq$, $>$ o $\geq$. Los bloques del índice del nivel más bajo se exploran o bien desde el valor más pequeño hasta v (para $<$ y $\leq$) o bien desde v hasta el valor más grande (para $>$ y $\geq$).

  El índice secundario proporciona punteros a los registros, pero para obtener los registros reales hay que extraer los registros usando los punteros. Este paso puede requerir una operación E/S por cada registro extraído, dado que los registros consecutivos pueden estar en diferentes bloques de disco. Si el número de registros extraídos es grande, el uso del índice secundario puede ser incluso más caro que la búsqueda lineal. Por tanto, el índice secundario sólo se debería usar si se seleccionan muy pocos registros.

13.3.4. Implementación de selecciones complejas

Hasta ahora sólo se han considerado condiciones de selección simples de la forma *A op B*, donde *op* es una operación igualdad o de comparación. Se revisarán a continuación predicados de selección más complejos.

- **Conjunción:** Una *selección conjuntiva* es una selección de la forma

$$\sigma_{\theta_1 \wedge \theta_2 \wedge \ldots \wedge \theta_n}(r)$$

- **Disyunción:** Una *selección disyuntiva* es una selección de la forma

$$\sigma_{\theta_1 \vee \theta_2 \vee \ldots \vee \theta_n}(r)$$

  Una condición disyuntiva se cumple mediante la unión de todos los registros que cumplen las condiciones individuales θ_i.
- **Negación:** El resultado de una selección $\sigma_{\neg\theta}(r)$ es el conjunto de tuplas de r para las que la condición θ se evalúa a falso. En ausencia de nulos, este conjunto es simplemente el conjunto de tuplas que no están en $\sigma_\theta(r)$.

Se puede implementar una operación selección con una conjunción o una disyunción de condiciones sencillas utilizando alguno de los siguientes algoritmos:

- **A8 (selección conjuntiva utilizando un índice).** Primero hay que determinar si para un atributo hay disponible algún camino de acceso en alguna de las condiciones simples. Si lo hay, cualquiera de los algoritmos de selección A2 hasta A7 puede recuperar los registros que cumplan esa condición. Se completa la operación mediante la comprobación en la memoria intermedia de que cada registro recuperado cumpla o no el resto de condiciones simples.

  Para reducir el coste, se elige un θ_i y uno de los algoritmos entre A1 y A7 para los que la combinación de resultados es el menor coste de $\sigma_{\theta_i}(r)$. El coste del algoritmo A8 está determinado por el coste del algoritmo elegido.
- **A9 (selección conjuntiva utilizando un índice compuesto).** Se podría disponer de un *índice compuesto* (es decir, un índice sobre varios atributos) apropiado para algunas selecciones conjuntivas. Si la selección especifica una condición de igualdad en dos o más atributos y existe un índice compuesto en estos campos con atributos combinados, entonces se podría buscar en el índice directamente. El tipo de índice determina cuál de los algoritmos A3, A4 o A5 se utilizará.
- **A10 (selección conjuntiva mediante la intersección de identificadores).** Otra alternativa para implementar la operación selección conjuntiva implica la utilización de punteros a registros o identificadores de registros. Este algoritmo necesita índices con punteros a registros en los campos involucrados por cada condición individual. De este modo se explora cada índice en busca de punteros cuyas tuplas cumplan alguna condición indi-

vidual. La intersección de todos los punteros recuperados forma el conjunto de punteros a tuplas que satisfacen la condición conjuntiva. Luego se usa el conjunto de punteros para recuperar los registros reales. Si no hubiera índices disponibles para algunas condiciones concretas entonces habría que comprobar el resto de condiciones de los registros recuperados.

El coste del algoritmo **A10** es la suma de los costes de las cada una de las exploraciones del índice más el coste de recuperar los registros en la intersección de las listas recuperadas de punteros. Este coste se puede reducir ordenando la lista de punteros y recuperando registros en orden. Por tanto, (1) todos los punteros a los registros de un bloque van juntos, y así todos los registros seleccionados en el bloque se pueden recuperar usando una única operación E/S, y (2) los bloques se leen ordenados, minimizando el movimiento del brazo del disco. El apartado 13.4 describe algunos algoritmos de ordenación.

- **A11 (selección disyuntiva mediante la unión de identificadores).** Si se disponen de caminos de acceso en todas las condiciones de la selección disyuntiva, se explora cada índice en busca de punteros cuyas tuplas cumplan una condición individual. La unión de todos los punteros recuperados proporciona el conjunto de punteros a todas las tuplas que cumplen la condición disyuntiva. Luego se utilizan estos punteros para recuperar los registros reales.

  Sin embargo, aunque solamente una de las condiciones no tenga un camino de acceso, se tiene que realizar una búsqueda lineal en la relación para encontrar todas las tuplas que cumplan la condición. Por tanto, aunque sólo exista una de estas condiciones en la disyunción, el método de acceso más eficiente es una exploración lineal, comprobando durante la búsqueda la condición disyuntiva en cada tupla.

La implementación de selecciones con condiciones negativas se deja como ejercicio (Ejercicio 13.10).

13.4. ORDENACIÓN

La ordenación de los datos juega un papel importante en los sistemas de bases de datos por dos razones. Primero, las consultas SQL pueden solicitar que los resultados sean ordenados. Segundo, e igualmente importante para el procesamiento de consultas, varias de las operaciones relacionales, como las reuniones, se pueden implementar eficientemente si las relaciones de entrada están ordenadas. Por este motivo se revisa la ordenación antes que la operación reunión en el Apartado 13.5.

Se puede conseguir la ordenación mediante la construcción de un índice en la clave de ordenación y utilizando luego ese índice para leer la relación de manera ordenada. No obstante, este proceso ordena la relación sólo *lógicamente* a través de un índice, en lugar de *físicamente*. Por tanto, la lectura de las tuplas de manera ordenada podría implicar acceder al disco para cada tupla. Por esta razón sería deseable ordenar las tuplas físicamente.

El problema de la ordenación se ha estudiado ampliamente para los casos en que la relación cabe completamente en memoria principal, y para el caso en el que la relación es mayor que la memoria. En el primer caso se utilizan técnicas de ordenación clásicas como la ordenación rápida (*quicksort*). Aquí se discute cómo tratar el segundo caso.

La ordenación de relaciones que no caben en memoria se llama **ordenación externa**. La técnica más utilizada para la ordenación externa es normalmente el algoritmo de **ordenación-mezcla externa**. A continuación se describe el algoritmo de ordenación-mezcla externa. Sea M el número de marcos de página en la memoria intermedia de la memoria principal (el número de bloques de disco cuyos contenidos se pueden alojar en la memoria intermedia de la memoria principal).

1. En la primera etapa, se crean varias *secuencias* ordenadas.

   i = 0;
   repeat
   leer M bloques o bien de la relación o bien del resto de la relación según el que tenga menor número de bloques;
   ordenar la parte en memoria de la relación;
   escribir los datos ordenados al archivo de secuencias S_i;
   i = i + 1;
   until el final de la relación

2. En la segunda etapa, las secuencias se *mezclan*. Supóngase que, por ahora, el número total de secuencias N es menor que M, así que se puede asignar un marco de página para cada secuencia y reservar espacio para guardar una página con el resultado. La etapa de mezcla se lleva a cabo de la siguiente manera:

   leer un bloque de cada uno de los N archivos S_i y guardarlos en una página de la memoria intermedia en memoria;
   repeat
   elegir la primera tupla (según el orden) de entre todas las páginas de la memoria intermedia;

escribir la tupla y suprimirla de la página de la memoria intermedia;
if la página de la memoria intermedia de alguna secuencia S_i está vacía **and** **not** fin-de-archivo(S_i) **then** leer el siguiente bloque de S_i y guardarlo en la página de la memoria intermedia;
until todas las páginas de la memoria intermedia estén vacías

El resultado de la etapa de mezcla es la relación ya ordenada. El archivo de salida se almacena en una memoria intermedia para reducir el número de operaciones de escritura en el disco. La operación anterior de mezcla es una generalización de la mezcla de dos vías utilizado por el algoritmo normal de ordenación-mezcla en memoria; éste mezcla *N* secuencias, por eso se llama **mezcla de n vías**.

En general, si la relación es mucho más grande que la memoria, se podrían generar *M* o más secuencias en la primera etapa y no sería posible asignar un marco de página para cada secuencia durante la etapa de mezcla. En este caso, se realiza la operación de mezcla en varios ciclos. Como hay suficiente memoria para $M - 1$ páginas de la memoria intermedia de entrada, cada mezcla puede tomar $M - 1$ secuencias como entrada.

El *ciclo* inicial se realiza como sigue. Se mezclan las $M - 1$ secuencias primeras (según se describió anteriormente) para obtener un única secuencia en el siguiente ciclo. Luego se mezclan de manera similar las siguientes $M - 1$ secuencias, continuando así hasta que todas las secuencias iniciales se hayan procesado. En este punto, el número de secuencias se ha reducido por un factor de $M - 1$. Si el número reducido de secuencias es todavía mayor o igual que *M*, se realiza otro ciclo con las secuencias creadas en el primer ciclo como entrada. Cada ciclo reduce el número de secuencias por un factor de $M - 1$. Se repiten estos ciclos tantas veces como sea necesario, hasta que el número de secuencias sea menor que *M*; momento en el que un último ciclo genera el resultado ordenado.

En la Figura 13.3 se ilustran los pasos de la ordenación-mezcla externa en una relación ficticia. Por motivos didácticos supóngase que solamente cabe una tupla en cada bloque ($f_r = 1$) y que la memoria puede contener como mucho tres marcos de página. Durante la etapa de mezcla se utilizan dos marcos de página como entrada y uno para la salida.

Calculemos cuántas transferencias de bloques se necesitan para la ordenación-mezcla externa. Sea b_r el número de bloques que contienen registros de la relación *r*. En la primera etapa, cada bloque de la relación se lee y se copia de nuevo, dando un total de $2b_r$ accesos a disco. El número inicial de secuencias es $\lceil b_r / M \rceil$. Puesto que el número de secuencias decrece en un factor de $M - 1$ en cada ciclo de la mezcla, el número total de ciclos requeridos viene dado por la expresión $\lceil \log_{M-1} (b_r / M) \rceil$. Cada uno de estos

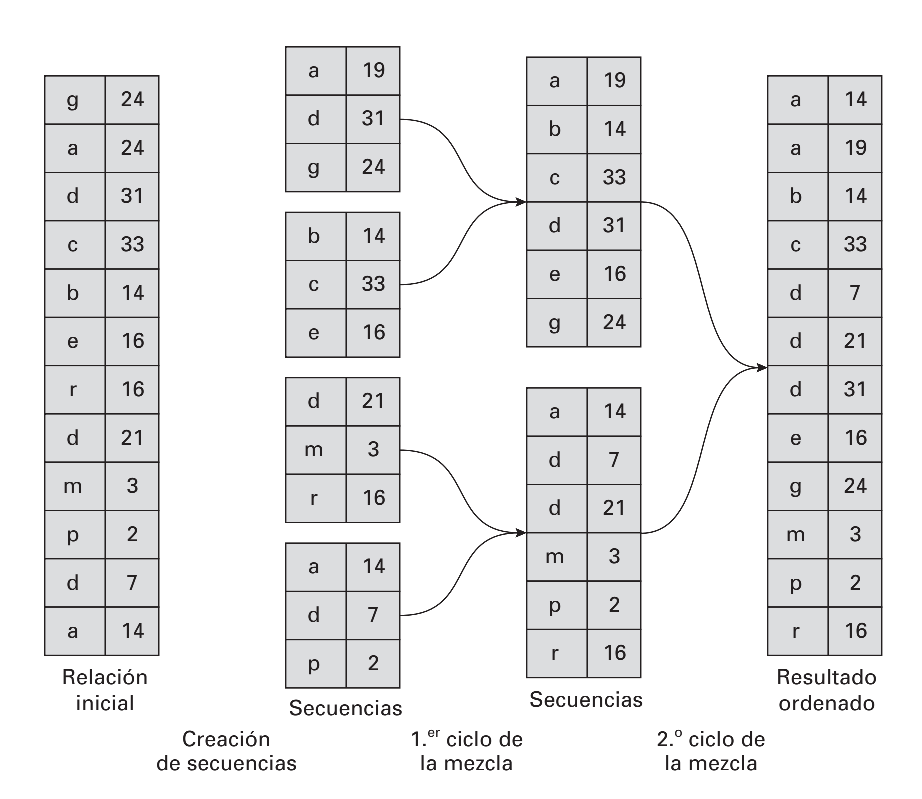


FIGURA 13.3. Ordenación externa utilizando ordenación-mezcla.

ciclos leen todos los bloques de la relación una vez y los copian al momento, con dos excepciones. Primero, el ciclo final puede producir la ordenación como resultado sin escribir el resultado en disco. Segundo, podría haber secuencias que ni lean ni copien durante un ciclo; por ejemplo, si hay M secuencias para mezclar en un ciclo, se leen $M – 1$ y se mezclan, para dejar la ejecución restante sin ser accedida durante este ciclo. Si se ignora el número (relativamente pequeño) de almacenamientos debido al último efecto, el número total de accesos a disco para la ordenación externa es

$$b_r (2 \lceil \log_{M-1} (b_r / M) \rceil + 1)$$

Aplicando esta ecuación al ejemplo de la Figura 13.3 se obtiene un total de 12 ∗ (4 + 1) = 60 bloques transferidos, como se puede comprobar en la figura. Obsérvese que este valor no incluye el coste de escribir el resultado final.

13.5. OPERACIÓN REUNIÓN

En este apartado se estudian varios algoritmos para calcular la reunión de relaciones y para analizar sus costes asociados.

Se utiliza la palabra **equirreunión** para hacer referencia a las reuniones de la forma $r \bowtie_{r.A=s.B} s$, donde A y B son atributos o conjuntos de atributos de las relaciones r y s respectivamente.

Utilizaremos como ejemplo la expresión

$$impositor \bowtie cliente$$

Se asume la siguiente información de catálogo acerca de las dos relaciones:

- Número de registros de *cliente*: $n_{cliente}$ = 10.000.
- Número de bloques de *cliente*: $b_{cliente}$ = 400.
- Número de registros de *impositor*: $n_{impositor}$ = = 5.000.
- Número de bloques de *impositor*: $b_{impositor}$ = 100.

13.5.1. Reunión en bucle anidado

En la Figura 13.4 se muestra un algoritmo sencillo para calcular la reunión zeta, $r \bowtie_\theta s$, de dos relaciones r y s. Este algoritmo se llama de **reunión en bucle anidado**, ya que básicamente consiste en un par de bucles **for** anidados. La relación r se denomina la **relación externa** y s la **relación interna** de la reunión, puesto que el bucle de r incluye al bucle de s. El algoritmo utiliza la notación $t_r \cdot t_s$, donde t_r y t_s son tuplas; $t_r \cdot t_s$ denota a la tupla construida mediante la concatenación de los valores de los atributos de las tuplas t_r y t_s.

```
for each tupla tr in r do begin
    for each tupla ts in s do begin
        comprobar que el par (tr, ts) satisface la condición θ de
        la reunión
        si la cumple, añadir tr · ts al resultado.
    end
end
```

FIGURA 13.4. Reunión en bucle anidado.

Al igual que el algoritmo de búsqueda lineal en archivos de la selección, el algoritmo de reunión en bucle anidado no necesita índices y se puede utilizar sin importar la condición de la reunión. La manera de extender el algoritmo para calcular la reunión natural es directa, puesto que la reunión natural se puede expresar como una reunión zeta seguida de la eliminación de los atributos repetidos mediante una proyección. El único cambio que se necesita es un paso adicional para borrar los atributos repetidos de la tupla $t_r \cdot t_s$ antes de añadirla al resultado.

El algoritmo de reunión en bucle anidado es costoso, ya que examina cada pareja de tuplas en las dos relaciones. Considérese el coste del algoritmo de reunión en bucle anidado. El número de pares de tuplas a considerar es $n_r * n_s$. Además, para cada registro en r, se tiene que realizar una exploración completa de s. En el peor de los casos, la memoria intermedia solamente puede contener un bloque de cada relación, necesitándose un total de $n_r * b_s + b_r$ accesos de bloques. En el mejor de los casos hay suficiente espacio para que las dos relaciones quepan en memoria, así que cada bloque se tendrá que leer solamente una vez; en consecuencia, sólo se necesitará acceder a $b_r + b_s$ bloques.

Si una de las relaciones cabe en memoria por completo, es útil usar esa relación como la relación más interna. De este modo se agiliza el procesamiento de la consulta significativamente, puesto que solamente será necesario leer una vez la relación del bucle más interno. Por lo tanto, si s es lo suficientemente pequeña para caber en la memoria principal, esta estrategia necesita solamente un total de $b_r + b_s$ accesos, el mismo coste que en el caso de las dos relaciones que caben en memoria.

Considérese ahora el caso de la reunión natural de *impositor* y *cliente*. Supóngase que, por el momento, no hay ningún índice en cualquiera de las relaciones y que no se desea crear un índice. Se pueden utilizar los bucles anidados para calcular la reunión; supóngase que *impositor* es la relación externa y que *cliente* es la relación interna de la reunión. Se tendrán que examinar 5.000 ∗ 10.000 = $50 \cdot 10^6$ pares de tuplas.

En el peor de los casos el número de accesos será de 5.000 ∗ 400 + 100 = 2.000.100 bloques. En la mejor de las situaciones, sin embargo, se tienen que leer ambas relaciones solamente una vez y realizar el cálculo. Este cálculo necesita a lo sumo 100 + 400 = 500 accesos a bloques, una mejora significativa sobre la situación en el peor de los casos. Si se hubiera utilizado *cliente* como la relación del bucle externo e *impositor* para el bucle interno, el coste en el peor de los casos de la última estrategia habría sido menor: 10.000 ∗ 100 + 400 = 1.000.400.

13.5.2. Reunión en bucle anidado por bloques

Si la memoria intermedia es demasiado pequeña para contener las dos relaciones por completo en memoria, todavía se puede lograr un mayor ahorro en los accesos a los bloques si se procesan las relaciones por bloques en lugar de por tuplas. El procedimiento de la Figura 13.5 muestra una variante de la reunión en bucle anidado, donde se empareja cada bloque de la relación interna con cada bloque de la relación externa. En cada par de bloques se empareja cada tupla de un bloque con cada tupla del otro bloque para generar todos los pares de tuplas. Al igual que antes, se añaden al resultado todas las parejas de tuplas que satisfacen la condición de la reunión.

La diferencia principal en coste entre la reunión en bucle anidado por bloques y la reunión en bucle anidado básica es que, en el peor de los casos, cada bloque de la relación interna *s* se lee solamente una vez por cada *bloque* de la relación externa, en lugar de una vez por cada *tupla* de la relación externa. De este modo, en el peor de los casos, habrá un total de $b_r * b_s + b_r$ accesos a bloques, donde b_r y b_s denotan respectivamente el número de bloques que contienen registros de *r* y *s*. Evidentemente, será más eficiente utilizar la relación más pequeña como la relación externa. En el mejor de los casos habrá que acceder a $b_r + b_s$ bloques.

Volvamos al ejemplo de calcular *impositor* ⋈ *cliente* pero utilizando ahora el algoritmo de reunión en bucle anidado por bloques. En el peor de los casos hay que leer una vez cada bloque de *cliente* por cada bloque de *impositor*. Así, en el peor caso, es necesario acceder un total de 100 ∗ 400 + 100 = 40.100 bloques. Este coste es una mejora importante frente a los 5.000 ∗ 400 + 100 = 2.000.100 accesos a bloques necesarios en el peor de los casos para la reunión en bucle anidado básica. Sin embargo el número de accesos a bloques en el mejor caso sigue siendo el mismo, es decir, 100 + 400 = 500.

El rendimiento de los procedimientos de bucle anidado y bucle anidado por bloques se puede mejorar aún más:

- Si los atributos de la reunión en una reunión natural o una equirreunión forman una clave de la relación interna, entonces el bucle interno puede finalizar tan pronto como se encuentre la primera correspondencia.
- En el algoritmo en bucle anidado por bloques, en lugar de utilizar bloques de disco como la unidad de bloqueo de la relación externa, se puede utilizar el mayor tamaño que quepa en memoria, mientras quede suficiente espacio para las memorias intermedias de la relación interna y la salida. En otras palabras, si la memoria tiene *M* bloques, se leen *M*-2 bloques de la relación externa de una vez, y cuando se lee cada bloque de la relación interna se reúne con los *M*-2 bloques de la relación externa. Este cambio reduce el número de exploraciones de la relación interna de b_r a $\lceil b_r/(M\text{-}2) \rceil$, donde b_r es el número de bloques de la relación externa. El coste total es $\lceil b_r/(M\text{-}2) \rceil * b_s + b_r$.
- Se puede explorar el bucle interno alternativamente hacia adelante y hacia atrás. Este método de búsqueda ordena las peticiones de bloques de disco de tal manera que los datos restantes en la memoria intermedia de la búsqueda anterior se reutilizan, reduciendo de este modo el número de accesos a disco necesarios.
- Si se dispone de un índice en un atributo de la reunión del bucle interno se pueden sustituir las búsquedas en archivos por búsquedas más eficientes en el índice. Esta optimización se describe en el Apartado 13.5.3.

13.5.3. Reunión en bucle anidado indexada

En una reunión en bucle anidado (Figura 13.4), si se dispone de un índice sobre el atributo de la reunión del

```
for each bloque Br of r do begin
    for each bloque Bs of s do begin
        for each tupla tr in Br do begin
            for each tupla ts in Bs do begin
                comprobar que el par (tr, ts) satisface la condición de la reunión
                si la cumple, añadir tr · ts al resultado.
            end
        end
    end
end
```

FIGURA 13.5. Reunión en bucle anidado por bloques.

bucle interno, se pueden sustituir las exploraciones de archivo por búsquedas en el índice. Para cada tupla t_r de la relación externa *r*, se utiliza el índice para buscar tuplas en *s* que cumplan la condición de reunión con la tupla t_r.

Este método de reunión se llama **reunión en bucle anidado indexada**; se puede utilizar cuando existen índices, así como cuando se crean índices temporales con el único propósito de evaluar la reunión.

La búsqueda de tuplas en *s* que cumplan las condiciones de la reunión con una tupla dada t_r es esencialmente una selección en *s*. Por ejemplo, considérese *impositor* ⋈ *cliente*. Supóngase que se tiene una tupla de *impositor* con *nombre-cliente* «Juan». Entonces, las tuplas relevantes en *s* son aquellas que satisfacen la selección «*nombre-cliente* = Juan».

El coste de la reunión en bucle anidado indexada se puede calcular como se indica a continuación. Para cada tupla en la relación externa *r* se realiza una búsqueda en el índice para *s* recuperando las tuplas apropiadas. En el peor de los casos sólo hay espacio en la memoria intermedia para una página de *r* y una página del índice. Por tanto, son necesarios b_r accesos a disco para leer la relación *r*, donde b_r denota el número de bloques que contienen registros de *r*. Para cada tupla de *r*, se realiza una búsqueda en el índice para *s*. Luego el coste de la reunión se puede calcular como $b_r + n_r \cdot c$, donde n_r es el número de registros de la relación *r* y *c* es el coste de una única selección en *s* utilizando la condición de la reunión. Ya se vio cómo estimar el coste del algoritmo de una única selección (posiblemente utilizando índices) cuyo cálculo proporciona el valor de *c*.

La fórmula del coste indica que, si hay índices disponibles en ambas relaciones *r* y *s*, normalmente es más eficiente usar como relación más externa aquella que tenga menos tuplas.

Por ejemplo, considérese una reunión en bucle anidado indexada de *impositor* ⋈ *cliente*, con *impositor* como la relación externa. Supóngase también que *cliente* tiene un índice de árbol B^+ primario en el atributo de la reunión *nombre-cliente,* que contiene 20 entradas en promedio por cada nodo del índice. Dado que *cliente* tiene 10.000 tuplas, la altura del árbol es 4, y será necesario un acceso más para encontrar el dato real. Como $n_{impositor}$ es 5.000, el coste total es 100 + 5.000 ∗ 5 = 25.100 accesos a disco. Este coste es menor que los 40.100 accesos necesarios para una reunión en bucle anidado por bloques.

```
pr:= dirección de la primera tupla de r;
ps:= dirección de la primera tupla de s;
while (ps ≠ null and pr ≠ null) do
     begin
          ts:= tupla a la que apunta ps;
          Ss:= {ts};
          hacer que ps apunte a la siguiente tupla de s;
          hecho:= falso;
          while (not hecho and ps ≠ null) do
               begin
                    ts':= tupla a la que apunta ps;
                    if (ts' [AtribsReunión] = ts [AtribsReunión])
                         then begin
                                   Ss:= Ss ∪ {ts'};
                                   hacer que ps apunte a la siguiente tupla de s;
                              end
                         else hecho:= cierto;
               end
          tr:= tupla a la que apunta pr;
          while (pr ≠ null and tr[AtribsReunión] < ts[AtribsReunión]) do
               begin
                    hacer que pr apunte a la siguiente tupla de r;
                    tr:= tupla a la que apunta pr;
               end
          while (pr ≠ null and tr[AtribsReunión] = ts[AtribsReunión]) do
               begin
                    for each ts in Ss do
                         begin
                              añadir ts ⋈ tr al resultado;
                         end
                    hacer que pr apunte a la siguiente tupla de r;
                    tr:= tupla a la que apunta pr;
               end
     end.
```

FIGURA 13.6. Reunión por mezcla.

13.5.4. Reunión por mezcla

El algoritmo de **reunión por mezcla** (también llamado algoritmo de **reunión por ordenación-mezcla**) se puede utilizar para calcular reuniones naturales y equirreuniones. Sean $r(R)$ y $s(S)$ las relaciones que vamos a utilizar para realizar la reunión natural, y sean $R \cap S$ sus atributos en común. Supóngase que ambas relaciones están ordenadas en los atributos de $R \cap S$. Entonces su reunión se puede calcular mediante un proceso muy parecido a la etapa de mezcla del algoritmo de ordenación-mezcla.

El algoritmo de reunión por mezcla se muestra en la Figura 13.6. En este algoritmo, *AtribsReunión* se refiere a los atributos en $R \cap S$ y, a su vez, $t_s \bowtie t_r$, donde t_r y t_s son las tuplas que tienen los mismos valores en *AtribsReunión*, denota la concatenación de los atributos de las tuplas, seguido de una proyección para no incluir los atributos repetidos. El algoritmo de reunión por mezcla asocia un puntero con cada relación. Al comienzo, estos punteros apuntan a la primera tupla de sus respectivas relaciones. Según avanza el algoritmo, el puntero se mueve a través de la relación. De este modo se leen en S_s un grupo de tuplas de una relación con el mismo valor en los atributos de la reunión. El algoritmo de la Figura 13.6 *necesita* que cada conjunto de tuplas S_s quepa en memoria principal; se examinará más tarde en este apartado extensiones del algoritmo que evitan este supuesto. Entonces, las tuplas correspondientes de la otra relación (si las hay) se leen y se procesan según se están leyendo.

En la Figura 13.7 se muestran dos relaciones que están ordenadas en su atributo de la reunión $a1$. Es instructivo recorrer los pasos del algoritmo de reunión por mezcla con las relaciones que se muestran en la figura.

Dado que las relaciones están ordenadas, las tuplas con el mismo valor en los atributos de la reunión aparecerán consecutivamente. De este modo solamente es necesario leer cada tupla en el orden una vez y, como

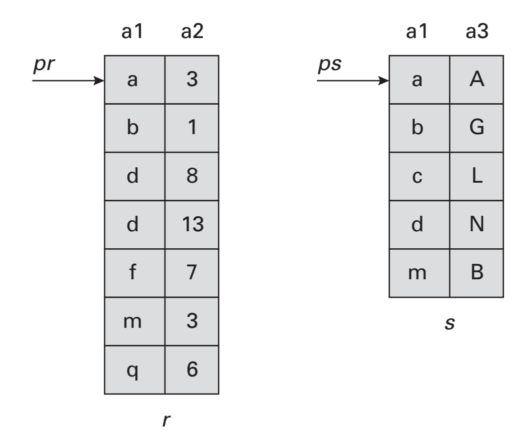


FIGURA 13.7. Relaciones ordenadas para la reunión por mezcla.

resultado, leer también cada bloque solamente una vez. Puesto que sólo se hace un ciclo en ambos archivos, el método de reunión por mezcla resulta eficiente; el número de accesos a bloques es igual a la suma de los bloques en los dos archivos, $b_r + b_s$.

Si alguna de las relaciones de entrada r o s no está ordenada según los atributos de la reunión, se pueden ordenar primero y luego utilizar el algoritmo de reunión por mezcla. El algoritmo de reunión por mezcla también se puede extender fácilmente desde las reuniones naturales al caso más general de las equirreuniones.

Supóngase que se aplica el esquema de reunión por mezcla al ejemplo de *impositor* $\bowtie$ *cliente*. Aquí el atributo de la reunión es *nombre-cliente*. Supóngase que las dos relaciones están ya ordenadas según el atributo de la reunión *nombre-cliente*. En este caso, la reunión por mezcla emplea un total de 400 + 100 = 500 accesos a bloques. Supóngase ahora que las relaciones no están ordenadas y que el tamaño de la memoria cae en el caso peor de tres bloques. Ordenar *cliente* emplea 400 $* (2\lceil log_2(400/3)\rceil + 1)$ o 6.800 transferencias de bloques, con 400 transferencias más para escribir el resultado. Del mismo modo, ordenar *impositor* lleva 100 $* (2\lceil log_2(100/3)\rceil + 1)$ o 1.300 transferencias de bloques, con 100 transferencias más para escribirlo. Así, el coste total si las relaciones no están ordenadas es de 9.100 transferencias de bloques y el tamaño de memoria es sólo 3 bloques.

Con un tamaño de memoria de 25 bloques, la ordenación de *cliente* emplea 400 $* (2\lceil log_2 (400/25)\rceil + 1)$ = 1.200 transferencias de bloques, mientras que la ordenación de *impositor* emplea 300 transferencias de bloques. Añadiendo el coste de escribir los resultados ordenados y volverlos a leer da un total de 2.500 transferencias de bloques si las relaciones no están ordenadas y el tamaño de la memoria es de 25 bloques.

Como se mencionó anteriormente, el algoritmo de reunión por mezcla de la Figura 13.6 necesita que el conjunto S_s de todas las tuplas con el mismo valor en los atributos de la reunión quepan en memoria principal. Este requisito se puede alcanzar normalmente, incluso si la relación s es grande. Si no se puede cumplir, se tiene que realizar entre S_s y las tuplas de r una reunión en bucle anidado por bloques con los mismos valores en los atributos de la reunión. Como resultado, el coste de la reunión por mezcla aumenta.

También es posible realizar una variación de la operación reunión por mezcla en tuplas desordenadas si existen índices secundarios en los dos atributos de la reunión. Así, se examinan los registros a través de los índices recuperándolos de manera ordenada. Sin embargo, esta variación presenta un importante inconveniente, puesto que los registros podrían estar diseminados a través de los bloques del archivo. Por tanto, cada acceso a una tupla podría implicar acceder a un bloque del disco, y esto es muy costoso.

Para evitar este coste se puede utilizar una técnica de reunión por mezcla híbrida que combinase índices

con reunión por mezcla. Supóngase que una de las relaciones está ordenada; la otra está desordenada, pero tiene un índice secundario de árbol B$^+$ en los atributos de la reunión. El **algoritmo híbrido de reunión por mezcla** fusiona la relación ordenada con las entradas hoja del índice secundario de árbol B$^+$. El archivo resultante contiene tuplas de la relación ordenada y direcciones de las tuplas de la relación desordenada. Luego se ordena el archivo resultante según las direcciones de las tuplas de la relación desordenada, permitiendo así una recuperación eficiente de las correspondientes tuplas, según el orden físico de almacenamiento, para completar la reunión. Se deja como ejercicio las extensiones de esta técnica para tratar el caso de dos relaciones desordenadas.

13.5.5. Reunión por asociación

Al igual que el algoritmo de reunión por mezcla, el algoritmo de reunión por asociación se puede utilizar para implementar reuniones naturales y equirreuniones. En el algoritmo de reunión por asociación se utiliza una función de asociación *h* para dividir las tuplas de ambas relaciones. La idea fundamental es dividir las tuplas de cada relación en conjuntos con el mismo valor de la función de asociación en los atributos de la reunión.

Se supone que

- *h* es una función de asociación que asigna a los *AtribsReunión* los valores $\{0, 1, ..., n_h\}$, donde los *AtribsReunión* denotan los atributos comunes de *r* y *s* utilizados en la reunión natural.
- $H_{r_0}, H_{r_1}, ..., H_{r_{n_h}}$ denotan las particiones de las tuplas de *r*, inicialmente todas vacías. Cada tupla $t_r \in r$ se pone en la partición H_{r_i}, donde $i = h(t_r[\textit{AtribsReunión}])$.
- $H_{s_0}, H_{s_1}, ..., H_{s_{n_h}}$ denotan las particiones de las tuplas de *s*, inicialmente todas vacías. Cada tupla $t_s \in s$ se pone en la partición H_{s_i}, donde $i = h(t_s[\textit{AtribsReunión}])$.

La función de asociación *h* debería de tener las «buenas» propiedades de aleatoriedad y uniformidad que se discutieron en el Capítulo 12. La división de las relaciones se muestra de manera gráfica en la Figura 13.8.

La idea que está detrás del algoritmo de reunión por asociación es la que sigue. Supóngase que una tupla de *r* y una tupla de *s* satisfacen la condición de la reunión; por tanto, tendrán el mismo valor en los atributos de la reunión. Si el valor se asocia con algún valor *i*, la tupla de *r* tiene que estar en H_{r_i} y la tupla de *s* en H_{s_i}. De este modo solamente es necesario comparar las tuplas de *r* en H_{r_i} con las tuplas de *s* en H_{s_i}; no es necesario compararlas con las tuplas de *s* de otra partición.

Por ejemplo, si *i* es una tupla de *impositor*, *c* una tupla de *cliente* y *h* una función de asociación en los atributos *nombre-cliente* de las tuplas, solamente se tiene que comprobar *i* y *c* si $h(c) = h(i)$. Si $h(c) \neq h(i)$,

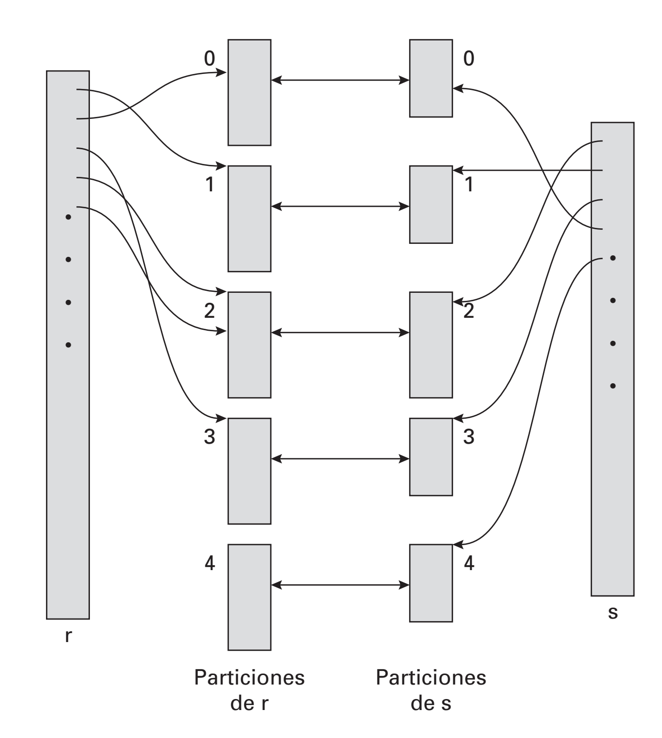


FIGURA 13.8. Particiones asociadas de relaciones.

entonces *c* e *i* tienen valores distintos en el *nombre-cliente*. Sin embargo, si $h(c) = h(i)$ hay que verificar que *c* e *i* tengan los mismos valores en los atributos de la reunión, ya que es posible que *c* e *i* tengan atributos *nombre-cliente* distintos con el mismo valor de la función de asociación.

En la Figura 13.9 se muestran los detalles del algoritmo de **reunión por asociación** para calcular la reunión natural de las relaciones *r* y *s*. Como en el algoritmo de reunión por mezcla, $t_r \bowtie t_s$ denota la concatenación de los atributos de las tuplas de t_r y t_s, seguido de la proyección para eliminar los atributos repetidos. Después de la división de las relaciones, el resto del código de reunión por asociación realiza una reunión en bucle anidado indexada separada en cada una de los partición pares *i*, con $i = 0, ..., n_h$. Para lograr esto, primero se **construye** un índice asociativo en cada H_{s_i} y luego se **prueba** (es decir, se busca en H_{s_i}) con las tuplas de H_{r_i}. La relación *s* es la **entrada para construir** y *r* es la **entrada para probar**.

El índice asociativo en H_{s_i} se construye en memoria, así que no es necesario acceder al disco para recuperar las tuplas. La función de asociación utilizada para construir este índice asociativo es distinta de la función de asociación *h* utilizada anteriormente, pero aún se aplica exclusivamente a los atributos de la reunión. A lo largo de la reunión en bucle anidado indexada, el sistema utiliza este índice asociativo para recuperar los registros que concuerden con los registros de la entrada para probar.

Las etapas de construcción y prueba necesitan solamente un único ciclo a través de las entradas para cons-

```
/* Partición de s */
for each tupla t_s in s do begin
    i := h(t_s [AtribsReunión]);
    H_si := H_si ∪ {t_s};
end
/* Partición de r */
for each tupla t_r in r do begin
    i := h(t_r [AtribsReunión]);
    H_ri := H_ri ∪ {t_r};
end
/* Realizar la reunión de cada partición */
for i: = 0 to max do begin
    leer H_si y construir un índice asociativo en memoria en él
    for each tupla t_r in H_ri do begin
        explorar el índice asociativo en H_si para localizar todas las tuplas t_s tales que t_s [AtribsReunión] = t_r [AtribsReunión]
        for each tupla t_s que concuerde in H_si do begin
            añadir t_r ⋈ t_s al resultado
        end
    end
end
```

FIGURA 13.9. Reunión por asociación.

truir y probar. La extensión del algoritmo de reunión por asociación para calcular equirreuniones más generales es directa.

Se tiene que elegir el valor de n_h lo suficientemente grande de tal modo que, para cada *i*, las tuplas en la partición H_{s_i} de la relación para construir, junto con el índice asociativo de la partición, quepan en la memoria. Pero no tienen por qué caber en memoria las particiones de la relación para probar. Claramente, será mejor utilizar la relación de entrada más pequeña como la relación para construir. Si el tamaño de la relación para construir es de b_s bloques, entonces para que cada una de las n_h particiones tengan un tamaño menor o igual que *M*, n_h debe ser al menos $\lceil b_s/M \rceil$. Con más exactitud, hay que tener en cuenta también el espacio adicional ocupado por el índice asociativo de la partición, incrementando n_h según corresponda. Por motivos de simplicidad muchas veces se ignora en estos análisis el requisito del espacio ocupado por el índice asociativo.

13.5.5.1. *División recursiva*

Si el valor de n_h es mayor o igual que el número de marcos de página de la memoria, la división de las relaciones no se puede hacer en un solo ciclo, puesto que no habría suficientes páginas para memorias intermedias. En lugar de eso, la división se tiene que hacer mediante la repetición de ciclos. En un ciclo se puede dividir la entrada en tantas particiones como marcos de página haya disponibles para utilizarlos como memorias intermedias de salida. Cada cajón generado por un ciclo se lee de manera separada y se divide de nuevo en el siguiente ciclo para crear particiones más pequeñas. La función de asociación utilizada en un ciclo es, por supuesto, diferente de la que se ha utilizado en el ciclo anterior. Se repite esta división de la entrada hasta que cada partición de la entrada para construir quepa en memoria. Esta división se llama **división recursiva**.

Una relación no necesita división recursiva si $M > n_h + 1$, o equivalentemente $M > (b_s/M) + 1$, que se simplifica (de manera aproximada) a $M > \sqrt{b_s}$. Por ejemplo, considerando un tamaño de la memoria de 12 MB, dividido en bloques de 4 KB; contendría un total de 3 K (3072) bloques. Se puede utilizar una memoria de este tamaño para dividir relaciones de hasta 3 K ∗ 3 K bloques de tamaño, que son 36 GB. Del mismo modo, una relación del tamaño de 1 GB necesita $\sqrt{256 \text{ K}}$ bloques, o alrededor de 2 MB, para evitar la división recursiva.

13.5.5.2. *Gestión de desbordamientos*

Se produce el **desbordamiento de una tabla de asociación** en la partición *i* de la relación para construir *s*, si el índice asociativo en H_{s_i} es más grande que la memoria principal. El desbordamiento de la tabla de asociación puede ocurrir si hay muchas tuplas en la relación para construir con los mismos valores en los atributos de la reunión, o si la función de asociación no tiene las propiedades de aleatoriedad y uniformidad. En cualquier caso, algunas de las particiones tendrán más tuplas que la media, mientras que otras tendrán menos; se dice entonces que la división está **sesgada**.

Se puede controlar parcialmente el sesgo mediante el incremento del número de particiones de tal manera que el tamaño esperado de cada partición (incluyendo el índice asociativo en la partición) sea algo menor que el tamaño de la memoria. Por consiguiente, el número de particiones se incrementa en un pequeño valor llamado **factor de escape**, que normalmente está alrededor del 20 por ciento del número de particiones calculadas con el método anterior.

Incluso si se es conservador con los tamaños de las particiones utilizando un factor de escape, todavía pueden ocurrir desbordamientos. Los desbordamientos de la tabla de asociación se pueden tratar mediante *resolución del desbordamiento* o *evitación del desborda-*

miento. La **resolución del desbordamiento** se realiza como sigue. Si H_{s_i} para cualquier i, resulta ser demasiado grande, se divide de nuevo en particiones más pequeñas utilizando una función de asociación distinta. Del mismo modo, también se divide H_{r_i} utilizando la nueva función de asociación, y solamente es necesario reunir las tuplas en las particiones concordantes.

Por el contrario, la **evitación del desbordamiento** realiza la división más cuidadosamente, de tal manera que el desbordamiento nunca se produce en la fase de construcción. La evitación de desbordamiento se implementa mediante la división de la relación para construir s inicialmente en muchas particiones pequeñas, para luego combinar algunas de estas particiones de tal manera que cada partición combinada quepa en memoria. Además, la relación para probar r se tiene que combinar de la misma manera que se combinan las particiones de s, sin importar los tamaños de H_{r_i}.

Las técnicas de resolución y evitación podrían fallar en algunas particiones, si un gran número de las tuplas en s tuvieran el mismo valor en los atributos de la reunión. En ese caso, en lugar de crear un índice asociativo en memoria y utilizar una reunión en bucle anidado para reunir las particiones, se pueden utilizar otras técnicas de reunión, tales como la reunión en bucle anidado por bloques, en esas particiones.

13.5.5.3. Coste de una reunión por asociación

Se considera ahora el coste de una reunión por asociación. El análisis supone que no hay desbordamiento de la tabla de asociación. Primero se considera el caso en el que no es necesaria una división recursiva. La división de las dos relaciones r y s reclama una lectura completa de ambas relaciones así como su posterior escritura. Este operación necesita $2(b_r + b_s)$ accesos a bloques, donde b_r y b_s denotan respectivamente el número de bloques que contienen registros de las relaciones r y s. Las fases de construcción y prueba leen cada una de las particiones una vez, empleando $b_r + b_s$ accesos adicionales. El número de bloques ocupados por las particiones podría ser ligeramente mayor que $b_r + b_s$ debido a que los bloques están parcialmente ocupados. El acceso a estos bloques llenos en parte puede añadir un gasto adicional de $2n_h$ a lo sumo, ya que cada una de las n_h particiones podría tener un bloque lleno parcialmente que se tiene que escribir y leer de nuevo. Así, el coste estimado para una reunión por asociación es

$$3(b_r + b_s) + 2n_h$$

La sobrecarga $2n_h$ es muy pequeña comparada con $b_r + b_s$ y se puede ignorar.

Consideremos ahora el caso en el que se necesita la división recursiva. Cada ciclo reduce el tamaño de cada una de las particiones por un factor esperado de $M - 1$; esta reducción del tamaño se repite hasta que cada partición tenga como mucho un tamaño de M bloques. Por tanto, el número esperado de ciclos necesarios para dividir s es $\lceil log_{M-1}(b_s) - 1 \rceil$. Puesto que cada bloque de s se lee y se escribe en cada ciclo, el número total de transferencias de bloques para dividir s es $2b_s \lceil log_{M-1}(b_s) - 1 \rceil$. Como el número de ciclos para dividir r es el mismo que el número de pasadas para dividir s, se obtiene la siguiente estimación del coste de la reunión:

$$2(b_r + b_s)\lceil \log_{M-1}(b_s) - 1 \rceil + b_r + b_s$$

Considérese, por ejemplo, la reunión *cliente* ⋈ *impositor*. Con un tamaño de memoria de 20 bloques, se puede dividir *impositor* en cinco partes, cada una de 20 bloques, con un tamaño tal que caben en memoria. Así, sólo es necesario un ciclo para la división. De la misma manera la relación *cliente* se divide en cinco particiones, cada una de tamaño 80. Si se ignora el coste de escribir los bloques parcialmente llenos, el coste es 3(100 + 400) = 1.500 transferencias de bloques.

Se puede mejorar la reunión por asociación si el tamaño de la memoria principal es grande. Cuando la entrada para construir se puede guardar por completo en memoria principal hay que poner n_h a 0; de este modo el algoritmo de reunión por asociación se ejecuta rápidamente, sin dividir las relaciones en archivos temporales y sin importar el tamaño de la entrada para probar. El coste estimado desciende a $b_r + b_s$.

13.5.5.4. Reunión por asociación híbrida

El algoritmo de **reunión por asociación híbrida** realiza otra optimización; es útil cuando los tamaños de la memoria son relativamente grandes pero no cabe toda la relación para construir en memoria. La fase de división del algoritmo de reunión por asociación necesita de un bloque de memoria como memoria intermedia para cada partición que se cree, más un bloque de memoria como memoria intermedia de entrada. Por tanto, se necesitan $n_h + 1$ bloques de memoria para dividir las dos relaciones. Si la memoria es más grande que $n_h + 1$, se puede utilizar el resto de la memoria ($M - n_h - 1$ bloques) para guardar en memorias intermedias la primera partición de la entrada para construir (esto es, H_{s_0}), así que no es necesario escribirla ni leerla de nuevo. Más aún, la función de asociación se diseña de tal manera que el índice asociativo en H_{s_0} quepa en $M - n_h - 1$ bloques, así que, al final de la división de s, H_{s_0} está en memoria por completo y se puede construir un índice asociativo en H_{s_0}.

Cuando r se divide de nuevo, las tuplas en H_{r_0} no se escriben en disco; en su lugar, según se van generando, el sistema las utiliza para examinar el índice asociativo residente en memoria de H_{s_0} y para generar las tuplas de salida de la reunión. Después de utilizarlas para la prueba, se descartan las tuplas, así que la partición H_{r_0} no ocupa ningún espacio en memoria. De este modo se ahorra un acceso de lectura y otro de escritura para cada bloque de H_{r_0} y H_{s_0}. Las tuplas en otras particiones se escriben de la manera usual para reunirlas más tarde. El ahorro de la reunión por asociación híbrida puede ser

importante si la entrada para construir es ligeramente mayor que la memoria.

Si el tamaño de la relación para construir es b_s, n_h es aproximadamente igual a b_s/M. Así, la reunión por asociación híbrida es más útil si $M >> b_s/M$, o si $M >> \sqrt{b_s}$, donde la notación >> significa *mucho más grande que*. Por ejemplo, supóngase que el tamaño del bloque es de 4 KB y que el tamaño de la relación para construir es de 1 GB. Entonces, el algoritmo híbrido de reunión por asociación es útil si el tamaño de la memoria es claramente mayor que 2 MB; las memorias de 100 MB o más son comunes en las computadoras de hoy en día.

Considérese de nuevo la reunión *cliente* ⋈ *impositor*. Con una memoria de 25 bloques de tamaño, se puede dividir *impositor* en cinco particiones, cada una de 20 bloques, y con la primera de las particiones de la relación para construir almacenada en memoria. Ocupa 20 bloques de memoria; un bloque se utiliza para la entrada y cuatro bloques más para guardar en memorias intermedias cada partición. De manera similar se divide la relación *cliente* en cinco particiones cada una de tamaño 80, la primera de las cuales se utiliza precisamente para probar, en lugar de escribirse y leerse de nuevo. Ignorando el coste de escribir los bloques parcialmente llenos, el coste es de 3(80 + 320) + 20 + 80 = 1.300 bloques transferidos, en lugar de las 1.500 transferencias de bloques sin la optimización de asociación híbrida.

13.5.6. Reuniones complejas

Las reuniones en bucle anidado y en bucle anidado por bloques son útiles sean cuales sean las condiciones de la reunión. Las otras técnicas de reunión son más eficientes que las reuniones en bucle anidado y sus variantes, aunque sólo se pueden utilizar con condiciones simples, tales como las reuniones naturales o las equirreuniones. Se pueden implementar reuniones con condiciones de la reunión más complejas, tales como conjunciones y disyunciones, utilizando las técnicas eficientes de la reunión mediante la aplicación de las técnicas desarrolladas en el Apartado 13.3.4 para el manejo de selecciones complejas.

Considérese la siguiente reunión con una condición conjuntiva:

$$r \bowtie_{\theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_n} s$$

Se pueden aplicar una o más de las técnicas de reunión descritas anteriormente en cada condición individual $r \bowtie_{\theta_1} s$, $r \bowtie_{\theta_2} s$, $r \bowtie_{\theta_3} s$, y así sucesivamente. Se calcula el resultado total de la reunión calculando primero el resultado de una de estas reuniones simples $r \bowtie_{\theta_i} s$; cada par de tuplas del resultado intermedio se compone de una tupla de *r* y otra de *s*. El resultado total de la reunión consiste en las tuplas del resultado intermedio que satisfacen el resto de condiciones

$$\theta_1 \wedge \dots \wedge \theta_{i-1} \wedge \theta_{i+1} \wedge \dots \wedge \theta_n$$

Estas condiciones se pueden ir comprobando según se generan las tuplas de $r \bowtie_{\theta i} s$.

Una reunión cuya condición es una disyunción se puede calcular como se indica a continuación. Considérese

$$r \bowtie_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n} s$$

La reunión se puede calcular como la unión de los registros de las reuniones $r \bowtie_{\theta_i} s$ individuales:

$$(r \bowtie_{\theta_1} s) \cup (r \bowtie_{\theta_2} s) \cup \dots \cup (r \bowtie_{\theta_n} s)$$

Los algoritmos para calcular la unión de relaciones se describen en el Apartado 13.6.

13.6. OTRAS OPERACIONES

Otras operaciones relacionales y operaciones relacionales extendidas —tales como eliminación de duplicados, proyección, operaciones sobre conjuntos, reunión externa y agregación— se pueden implementar según se describe en los Apartados 13.6.1 al 13.6.5.

13.6.1. Eliminación de duplicados

Se puede implementar fácilmente la eliminación de duplicados utilizando la ordenación. Las tuplas idénticas aparecerán consecutivas durante la ordenación, pudiéndose eliminar todas las copias menos una. Con la ordenación-mezcla externa, se pueden eliminar los duplicados mientras se crea una secuencia antes de que ésta se escriba en el disco, reduciendo así el número de bloques transferidos. El resto de duplicados se pueden suprimir durante la etapa de reunión/mezcla, así que el resultado final estará libre de repeticiones. El coste estimado en el peor de los casos para la eliminación de duplicados es el mismo que el coste estimado en el peor caso para la ordenación de una relación.

Se puede implementar también la eliminación de duplicados utilizando la asociación de una manera similar al algoritmo de reunión por asociación. Primero, se divide la relación basándose en una función de asociación en la tupla entera. Luego, se lee cada partición, y se construye un índice asociativo en memoria. Mientras se construye el índice asociativo se inserta una tupla solamente si no estaba ya presente. En otro caso se descarta. Después de que todas las tuplas de la relación se hayan

procesado, las tuplas en el índice asociativo se escriben al resultado. El coste estimado es el mismo que el coste de procesar (división y lectura de cada partición) la relación para construir en una reunión por asociación.

Debido al coste relativamente alto de la eliminación de duplicados, SQL requiere una petición explícita del usuario para suprimir los duplicados; en caso contrario se conservan los duplicados.

13.6.2. Proyección

Se puede implementar la proyección fácilmente realizando la proyección de cada tupla, pudiendo originar una relación con registros repetidos y suprimiendo después los registros duplicados. La eliminación de duplicados se puede llevar a cabo según se ha descrito en el Apartado 13.6.1. Si los atributos de la lista de proyección incluyen una clave de la relación, no se producirán duplicados; por tanto no será necesario eliminarlos. La proyección generalizada (discutida en el Apartado 3.3.1) se puede implementar de la misma manera que las proyecciones.

13.6.3. Operaciones sobre conjuntos

Se pueden implementar las operaciones *unión, intersección* y *diferencia de conjuntos* ordenando primero ambas relaciones y examinando después cada relación para producir el resultado. En $r \cup s$, cuando una exploración concurrente en ambas relaciones descubre la misma tupla en los dos archivos, solamente se conserva una de las tuplas. Por otra parte, el resultado de $r \cap s$ contendrá únicamente aquellas tuplas que aparezcan en ambas relaciones. De la misma manera se implementa la *diferencia de conjuntos*, $r - s$, guardando aquellas tuplas de r que estén ausentes en s.

Para todas estas operaciones solamente se necesita una exploración en cada relación de entrada, así el coste es $b_r + b_s$. Si las relaciones no están ordenadas inicialmente, hay que incluir el coste de la ordenación. Se puede utilizar cualquier otro orden en la evaluación de la operación de conjuntos, siempre que las dos entradas tengan la misma ordenación.

La asociación proporciona otra manera de implementar estas operaciones sobre conjuntos. El primer paso en cada caso es dividir las dos relaciones utilizando la misma función de asociación y de este modo crear las particiones $H_{r_0}, \ldots, H_{r_{n_h}}$ y $H_{s_0}, \ldots, H_{s_{n_h}}$. Lo siguiente se realiza en cada partición $i = 0, 1, \ldots, n_h$.

- $r \cup s$
  1. Construir un índice asociativo en memoria en H_{r_i}.
  2. Añadir las tuplas en H_{s_i} al índice asociativo solamente si no estaban ya presentes.
  3. Añadir las tuplas del índice asociativo al resultado.
- $r \cap s$
  1. Construir un índice asociativo en memoria en H_{r_i}.
  2. Para cada tupla en H_{s_i} probar el índice asociativo y pasar la tupla al resultado únicamente si ya estaba presente en el índice.
- $r - s$
  1. Construir un índice asociativo en memoria en H_{r_i}.
  2. Para cada tupla en H_{s_i} probar el índice asociativo y, si la tupla está presente en el índice, suprimirla del índice asociativo.
  3. Añadir las tuplas restantes del índice asociativo al resultado.

13.6.4. Reunión externa

Recordemos las *operaciones de reunión externa* que se describieron en el Apartado 3.3.3. Por ejemplo, la reunión externa por la izquierda *cliente* ⟕ *impositor* contiene la reunión de *cliente* e *impositor* y adicionalmente, para cada tupla t de *cliente* que no concuerde con alguna tupla en *impositor* (es decir, donde no esté *nombre-cliente* en *impositor*), se añade la siguiente tupla t_1 al resultado. Para todos los atributos en el esquema de *cliente*, la tupla t_1 tiene los mismos valores que la tupla t. El resto de atributos (del esquema de *impositor*) de la tupla t_1 contienen el valor nulo.

Se pueden implementar las operaciones de reunión externa empleando una de las dos estrategias siguientes.

1. Calcular la reunión correspondiente, y luego añadir más tuplas al resultado de la reunión hasta obtener la reunión externa resultado. Considérese la operación de reunión externa por la izquierda y dos relaciones: $r(R)$ y $s(S)$. Para evaluar $r ⟕_\theta s$, se calcula primero $r \bowtie_\theta s$ y se guarda este resultado como la relación temporal q_1. A continuación se calcula $r - \Pi_R(q_1)$, que produce las tuplas de r que no participaron en la reunión. Se puede utilizar cualquier algoritmo para calcular las reuniones. Luego se rellenan cada una de estas tuplas con valores nulos en los atributos de s y se añaden a q_1 para obtener el resultado de la reunión externa.

   La operación reunión externa por la derecha $r ⟖_\theta s$ es equivalente a $s ⟕_\theta r$ y, por tanto, se puede implementar de manera simétrica a la reunión externa por la izquierda. También se puede implementar la operación reunión externa completa $r ⟗_\theta s$ calculando la reunión $r \bowtie_\theta s$ y añadiendo entonces tuplas adicionales de operaciones reunión externa por la izquierda y por la derecha, al igual que antes.
2. Modificar los algoritmos de la reunión. Así, es fácil extender los algoritmos de reunión en bucle

anidado para calcular la reunión externa por la izquierda. Las tuplas de la relación externa que no concuerdan con ninguna tupla de la relación interna se escriben en la salida después de haber sido completadas con valores nulos. Sin embargo, es difícil extender la reunión en bucle anidado para calcular la reunión externa completa.

Las reuniones externas naturales completas y las reuniones externas con una condición de equirreunión, se pueden calcular mediante extensiones de los algoritmos de reunión por mezcla y reunión por asociación. La reunión por mezcla se puede extender para realizar la reunión externa completa como se muestra a continuación. Cuando se está produciendo la mezcla de las dos relaciones, las tuplas de la relación que no encajan con ninguna tupla de la otra relación se pueden completar con valores nulos y escribirse en la salida. Del mismo modo, se puede extender la reunión por mezcla para calcular las reuniones externas por la izquierda y por la derecha mediante la copia de las tuplas que no concuerden (rellenadas con valores nulos) desde solamente una de las relaciones. Puesto que las relaciones están ordenadas, es fácil detectar si una tupla coincide o no con alguna de las tuplas de la otra relación. Por ejemplo, cuando se hace una reunión por mezcla de *cliente* e *impositor*, las tuplas se leen según el orden de *nombre-cliente*, siendo fácil de comprobar para cada tupla si hay alguna tupla coincidente.

El coste estimado para implementar reuniones externas utilizando el algoritmo de reunión por mezcla es el mismo que para la correspondiente reunión. La única diferencia está en el tamaño del resultado y, por tanto, en los bloques transferidos para copiarlos que no se han tenido en cuenta en las estimaciones anteriores del coste.

La extensión del algoritmo de reunión por asociación para calcular reuniones externas se deja como ejercicio (Ejercicio 13.11).

13.6.5. Agregación

Retomemos el operador de agregación $\mathcal{G}$ discutido en el Apartado 3.3.2. Por ejemplo, la operación

$$_{nombre\text{-}sucursal}\,\mathcal{G}\,_{\mathbf{sum}\,(saldo)}\,(cuenta)$$

agrupa tuplas de *cuenta* por sucursal y calcula el saldo total de todas las cuentas en cada sucursal.

La operación agregación se puede implementar de una manera parecida a la eliminación de duplicados. Se utiliza la ordenación o la asociación, al igual que se hizo para la supresión de duplicados, pero basándose ahora en la agrupación de atributos (*nombre-sucursal* en el ejemplo anterior). Sin embargo, en lugar de eliminar las tuplas con el mismo valor en los atributos de la agrupación, se reúnen en grupos y se aplican las operaciones agregación en cada grupo para obtener el resultado.

El coste estimado para la implementación de la operación agregación es el mismo coste de la eliminación de duplicados para las funciones de agregación como **min**, **max**, **sum**, **count** y **avg**.

En lugar de reunir todas las tuplas en grupos y aplicar entonces las funciones de agregación, se pueden implementar las funciones de agregación **min**, **max**, **sum**, **count** y **avg** sobre la marcha según se construyen los grupos. Para el caso de **sum**, **min** y **max,** cuando se encuentran dos tuplas del mismo grupo se sustituyen por una sola tupla con **sum**, **min** o **max**, respectivamente, de las columnas que se están agregando. Para la operación **count**, se mantiene una cuenta incremental para cada grupo con tuplas descubiertas. Por último, la operación **avg** se implementa calculando la suma y la cuenta de valores sobre la marcha, para dividir finalmente la suma entre la cuenta para obtener la media.

Si todas las tuplas del resultado caben en memoria, las implementaciones basadas en ordenación y las basadas en asociación no necesitan escribir ninguna tupla en disco. Según se leen las tuplas se pueden insertar en un estructura ordenada de árbol o en un índice asociativo. Así, cuando se utilizan las técnicas de agregación sobre la marcha, solamente es necesario almacenar una tupla para cada uno de los grupos. Por tanto, la estructura ordenada de árbol o el índice asociativo van a caber en memoria y se puede procesar la agregación con sólo b_r transferencias de bloques, en lugar de las $3b_r$ transferencias que se necesitarían de otra manera.

13.7. EVALUACIÓN DE EXPRESIONES

Hasta aquí se ha estudiado cómo llevar a cabo operaciones relacionales individuales. Ahora se considera cómo evaluar una expresión que contiene varias operaciones. La manera evidente de evaluar una expresión es simplemente evaluar una operación a la vez en un orden apropiado. El resultado de cada evaluación se **materializa** en una relación temporal para su inmediata utilización. Un inconveniente de esta aproximación es la necesidad de construir relaciones temporales, que (a menos que sean pequeñas) se tienen que escribir en disco. Un enfoque alternativo es evaluar varias operaciones de manera simultánea en un **cauce**, con los resultados de una operación pasados a la siguiente, sin la necesidad de almacenar relaciones temporales.

En los Apartados 13.7.1 y 13.7.2 se consideran ambos enfoques, *materialización* y *encauzamiento*. Se verá que el coste de estos enfoques puede diferir sustancialmen-

te, pero también que existen casos donde sólo la materialización es posible.

13.7.1. Materialización

Intuitivamente es fácil de entender cómo evaluar una expresión observando una representación gráfica de la expresión en un **árbol de operadores**. Considérese la expresión

$$\Pi_{nombre\text{-}cliente} (\sigma_{saldo < 2500} (cuenta) \bowtie cliente)$$

que se muestra en la Figura 13.10.

Si se aplica el enfoque de la materialización, se comienza por las operaciones de la expresión de nivel más bajo (al fondo del árbol). En el ejemplo solamente hay una de estas operaciones; la operación selección en *cuenta*. Las entradas de las operaciones de nivel más bajo son las relaciones de la base de datos. Se ejecutan estas operaciones utilizando los algoritmos ya estudiados, almacenando sus resultados en relaciones temporales. Luego se utilizan estas relaciones temporales para ejecutar las operaciones del siguiente nivel en el árbol, cuyas entradas son ahora o bien relaciones temporales o bien relaciones almacenadas en la base de datos. En este ejemplo, las entradas de la reunión son la relación *cliente* y la relación temporal producida por la selección en *cuenta*. Ahora se puede evaluar la reunión creando otra relación temporal.

Repitiendo este proceso se calcularía finalmente la operación en la raíz del árbol, obteniendo el resultado final de la expresión. En el ejemplo se consigue el resultado final mediante la ejecución de la operación proyección de la raíz utilizando como entrada la relación temporal creada por la reunión.

Una evaluación como la descrita se llama **evaluación materializada**, puesto que los resultados de cada operación intermedia se crean (materializan) para utilizarse a continuación en la evaluación de las operaciones del siguiente nivel.

El coste de una evaluación materializada no es simplemente la suma de los costes de las operaciones involucradas. Cuando se calcularon los costes estimados de los algoritmos se ignoró el coste de escribir el resultado de la operación en disco. Para calcular el coste de evaluar una expresión como la que se ha hecho hay que añadir los costes de todas las operaciones, incluyendo el coste de escribir los resultados intermedios en disco. Supóngase que los registros del resultado se acumulan en una memoria intermedia y que cuando ésta se llena, los registros se escriben en el disco. El coste de copiar el resultado se puede estimar en n_r / f_r, donde n_r es el número aproximado de tuplas de la relación resultado y f_r es el *factor de bloqueo* de la relación resultado, es decir, el número de registros de r que caben en un bloque.

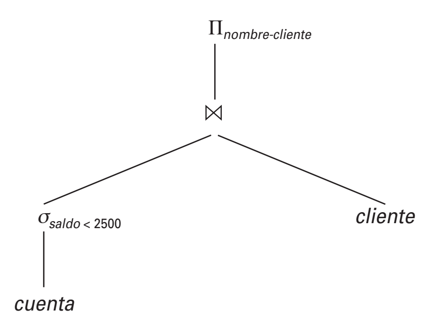


FIGURA 13.10. Representación gráfica de una expresión.

La **memoria intermedia doble** (usando dos memorias intermedias, una donde progresa la ejecución del algoritmo mientras que la otra se está copiando) permite que el algoritmo se ejecute más rápidamente mediante la ejecución en paralelo de acciones en la UCP con acciones de E/S.

13.7.2. Encauzamiento

Se puede mejorar la eficiencia de la evaluación de la consulta mediante la reducción del número de archivos temporales que se producen. Se lleva a cabo esta reducción con la combinación de varias operaciones relacionales en un *encauzamiento* de operaciones, en el que se pasan los resultados de una operación a la siguiente operación del encauzamiento. Esta evaluación, como se ha descrito, se denomina **evaluación encauzada**. La combinación de operaciones en un encauzamiento elimina el coste de leer y escribir relaciones temporales.

Por ejemplo, considérese la reunión de un par de relaciones seguida de una proyección ($\Pi_{a1,a2} (r \bowtie s)$). Si se aplicara la materialización, la evaluación implicaría la creación de una relación temporal para guardar el resultado de la reunión y la posterior lectura del resultado para realizar la proyección. Estas operaciones se pueden combinar como sigue. Cuando la operación reunión genera una tupla del resultado, se pasa inmediatamente esa tupla al operador de proyección para su procesamiento. Mediante la combinación de la reunión y de la proyección, se evita la creación de resultados intermedios, creando en su lugar el resultado final directamente.

13.7.2.1. Implementación del encauzamiento

Se puede implementar el encauzamiento construyendo una única y compleja operación que combine las operaciones que constituyen el encauzamiento. Aunque este enfoque aproximación podría ser factible en muchas situaciones, es deseable en general usar de nuevo el código en operaciones individuales en la construcción del encauzamiento. Por lo tanto, cada operación del encauzamiento se modela como un proceso aislado o una hebra en el sistema, que toma un flujo de tuplas de sus entradas encauzadas y produce un flujo de tuplas como salida. Para cada pareja de operaciones adyacentes en el encauzamiento se crea una memoria interme-

dia para guardar las tuplas que se envían de una operación a la siguiente.

En el ejemplo de la Figura 13.10, las tres operaciones se pueden situar en un encauzamiento, en el que los resultados de la selección se pasan a la reunión según se generan. Por su parte, los resultados de la reunión se envían a la proyección según se van generando. Así, los requisitos de memoria son bajos, ya que los resultados de una operación no se almacenan por mucho tiempo. Sin embargo, como resultado del encauzamiento, las entradas de las operaciones no están disponibles todas a la vez para su procesamiento.

Los encauzamientos se pueden ejecutar de alguno de los siguientes modos:

1. Bajo demanda
2. Desde los productores

En un **encauzamiento bajo demanda**, el sistema reitera peticiones de tuplas desde la operación de la cima del encauzamiento. Cada vez que una operación recibe una petición de tuplas, calcula la siguiente tupla (o tuplas) a devolver y la envía. Si las entradas de la operación no están encauzadas, la(s) siguiente(s) tupla(s) a devolver se calcula(n) de las relaciones de entrada mientras se lleva cuenta de lo que se ha remitido hasta el momento. Si alguna de sus entradas está encauzada, la operación también hace peticiones de tuplas desde sus entradas encauzadas. Así, utilizando las tuplas recibidas en sus entradas encauzadas, la operación calcula sus tuplas de salida y las envía hasta su padre.

En un **encauzamiento por los productores**, los operadores no esperan a que se produzcan peticiones para producir tuplas, en su lugar generan las tuplas **impacientemente**. Cada operación del fondo del encauzamiento genera continuamente tuplas de salida y las ponen en su memoria intermedia de salida hasta que se llena. Una operación en cualquier otro nivel del encauzamiento obtiene sus tuplas de entrada de un nivel inferior del encauzamiento hasta llenar su memoria intermedia de salida. Una vez que la operación ha utilizado una tupla de su entrada encauzada, la elimina de ella. En cualquier caso, una vez que la memoria intermedia de salida esté llena, la operación espera hasta que su operación padre elimine las tuplas de la memoria intermedia para hacer más espacio a nuevas tuplas. En este momento, la operación genera más tuplas hasta que se llene la memoria intermedia de nuevo. Este proceso se repite por una operación hasta que se hayan generado todas las tuplas de salida.

El sistema necesita cambiar de una operación a otra solamente cuando se llena una memoria intermedia de salida o cuando una memoria intermedia de entrada está vacía y se necesitan más tuplas para generar las tuplas de salida. En un sistema de procesamiento paralelo, las operaciones del encauzamiento se pueden ejecutar concurrentemente en distintos procesadores (véase el Capítulo 20).

Se puede imaginar el encauzamiento por los productores como una **inserción** de datos de abajo hacia arriba en el árbol de operaciones, mientras que el encauzamiento bajo demanda se puede pensar como una **extracción** de datos desde la cima del árbol de operaciones. Mientras que las tuplas se generan *impacientemente* en el encauzamiento por los productores, se generan de forma **perezosa**, bajo demanda, en el encauzamiento bajo demanda.

Cada operación en un encauzamiento bajo demanda se puede implementar como un **iterador**, que proporciona las siguientes funciones: abrir(), siguiente() y cerrar(). Después de una llamada a abrir(), cada llamada a siguiente() devuelve la siguiente tupla de salida de la operación. La implementación de la operación vigente produce llamadas a siguiente() desde sus entradas, cuando necesita tuplas de entrada. La función cerrar() comunica al iterador que no necesita más tuplas. De este modo, el iterador mantiene el **estado** de su ejecución entre las llamadas, de tal manera que las sucesivas peticiones siguiente() reciben sucesivas tuplas resultado.

Por ejemplo, en un iterador que implemente la operación selección usando la búsqueda lineal, la operación abrir() inicia una exploración de archivo y el estado del iterador registra el punto en el que el archivo se ha explorado. Cuando se llama a la función siguiente() la exploración del archivo continúa a continuación del punto anterior; cuando se encuentre la siguiente tupla que cumpla la selección al explorar el archivo, se devuelve la tupla después de almacenar el punto donde se encontró en el estado del iterador. Una operación abrir() del iterador de reunión por mezcla abriría sus entradas y, si aún no están ordenadas, también las ordenaría. En las llamadas a siguiente() se devolvería el siguiente par de tuplas coincidentes. La información de estado consistiría en volver a la posición en que se ha explorado cada entrada.

Los detalles de la implementación de iteradores se dejan para completar en el Ejercicio 13.12. El encauzamiento bajo demanda se utiliza normalmente más que el encauzamiento por los productores, ya que es más fácil de implementar.

13.7.2.2. Algoritmos de evaluación para el encauzamiento

Considérese una operación reunión cuya entrada del lado izquierdo está encauzada. Puesto que está encauzada, no toda la entrada está disponible al mismo tiempo para el procesamiento de la operación reunión. Al no estar disponible, se limita la elección del algoritmo de reunión a emplear. Por ejemplo, no se puede usar la reunión por mezcla si las entradas no están ordenadas, puesto que no es posible ordenar la relación hasta que todas las tuplas estén disponibles; así, en efecto, se convierte el encauzamiento en materialización. Sin embargo, se puede utilizar la reunión en bucle anidado inde-

xada: Según se reciben las tuplas por el lado izquierdo de la reunión se pueden utilizar para indexar el lado derecho de la relación y para generar las tuplas resultado de la reunión. Este ejemplo ilustra que las elecciones respecto del algoritmo a utilizar para una operación y las elecciones respecto del encauzamiento no son independientes.

Las restricciones en los algoritmos de evaluación que se pueden utilizar es un factor determinante del encauzamiento. Como resultado, a pesar de las aparentes ventajas del encauzamiento, hay casos donde la materialización alcanza un coste total menor. Supóngase que se desea realizar la reunión de *r* y *s*, y que la entrada *r* está encauzada. Si se utiliza una reunión en bucle anidado indexada para llevar a cabo el encauzamiento, podría ser necesario un acceso a disco para cada tupla de la relación de entrada encauzada. El coste de esta técnica es $n_r * AA_i$, donde AA_i es la altura de índice en *s*. Con la materialización, el coste de copiar *r* sería b_r. Con una técnica de reunión, como la reunión por asociación, podría ser posible realizar la reunión con un coste aproximado de $3(b_r + b_s)$. Si n_r es realmente mayor que $4b_r + 3b_s$, la materialización sería más económica.

El uso eficiente del encauzamiento necesita la utilización de algoritmos de evaluación que puedan generar tuplas de salida según se están recibiendo tuplas por las entradas de la operación. Se pueden distinguir dos casos:

1. Solamente una de las entradas de la reunión está encauzada.
2. Las dos entradas de la reunión están encauzadas.

Si únicamente una de las entradas de la reunión está encauzada, la reunión en bucle anidado indexada es la elección más natural. Si las tuplas de entrada encauzadas están ordenadas según los atributos de la reunión y la condición de la reunión es una equirreunión, también se puede emplear la reunión por mezcla. La reunión por asociación híbrida se puede utilizar con la entrada encauzada como la relación para probar. Sin embargo, las tuplas que no están en la primera partición se enviarán a la salida solamente después de que la relación de entrada encauzada se reciba por completo. La reunión por asociación híbrida es útil si las entradas no encauzadas caben completamente en memoria, o si al menos la mayoría de las entradas caben en memoria.

Si ambas entradas están encauzadas, la elección de los algoritmos de reunión está más limitada. Si ambas entradas están ordenadas en los atributos de la reunión y la condición de la reunión es una equirreunión, entonces se puede usar la reunión por mezcla. Otra técnica alternativa es la **reunión encauzada**, que se muestra en la Figura 13.11. El algoritmo supone que las tuplas de entrada de ambas relaciones, *r* y *s*, están encauzadas. Las tuplas disponibles de ambas relaciones se dejan listas para su procesamiento en una cola simple. Así mismo, se generan unas entradas de la cola especiales, llamadas Fin_r y Fin_s, que sirven como marcas de fin de archivo y que se insertan en la cola después de que se hayan generado todas las tuplas de *r* y *s* (respectivamente). Para una evaluación eficaz se deberían construir los índices apropiados en las relaciones *r* y *s*. Según se añaden tuplas a *r* y a *s* se deben mantener los índices actualizados.

```
hecho_r := falso;
hecho_s := falso;
r := ∅;
s := ∅;
resultado := ∅;
while not hecho_r or not hecho_s do
    begin
        if la cola está vacía then esperar hasta que la cola no esté vacía;
        t := entrada de la cima de la cola;
        if t = Fin_r then hecho_r := cierto
            else if t = Fin_s then hecho_s := cierto
                else if t es de la entrada r
                    then
                        begin
                            r := r ∪ {t};
                            resultado := resultado ∪ ({t} ⋈ s);
                        end
                    else /* t es de la entrada s */
                        begin
                            s := s ∪ {t};
                            resultado := resultado ∪ (r ⋈ {t});
                        end
end
```

FIGURA 13.11. Algoritmo de reunión encauzada.

13.8. RESUMEN

- La primera acción que el sistema debe realizar en una consulta es traducirla en su formato interno, que (para sistemas de bases de datos relacionales) está basado normalmente en el álgebra relacional. En el proceso de generación del formato interno de la consulta, el analizador comprueba la sintaxis, verifica que los nombres de relación que figuran en la consulta son nombres de relaciones de la base de datos, etcétera. Si la consulta se expresó en términos de una vista, el analizador sustituye todas las referencias al nombre de la vista con su expresión del álgebra relacional para calcularla.
- Dada una consulta, generalmente hay diversos métodos para calcular la respuesta. Es responsabilidad del sistema transformar la consulta proporcionada por el usuario en otra consulta equivalente que se pueda ejecutar de un modo más eficiente. El Capítulo 14 estudia la optimización de consultas.
- Se pueden procesar consultas que impliquen selecciones sencillas mediante una búsqueda lineal, una búsqueda binaria o utilizando índices. Así mismo se pueden manejar selecciones más complejas mediante uniones e intersecciones de los resultados de selecciones simples.
- Se pueden ordenar relaciones que sean más grandes que la memoria utilizando el algoritmo de ordenación-mezcla externa.
- Las consultas que impliquen una reunión natural se pueden procesar de varias maneras, dependiendo de la disponibilidad de índices y del tipo de almacenamiento físico utilizado para las relaciones.
- Si la reunión resultante es casi tan grande como el producto cartesiano de las dos relaciones, una estrategia de *reunión en bucle anidado por bloques* podría ser ventajosa.
- Si hay índices disponibles, se puede utilizar *la reunión en bucle anidado indexada*.
- Si las relaciones están ordenadas, sería deseable una *reunión por mezcla*. Además, podría ser útil ordenar una relación antes que calcular una reunión (para permitir el uso de una estrategia de reunión por mezcla).
- El algoritmo de *reunión por asociación* divide la relación en varias particiones, de tal manera que cada partición de una de las relaciones quepa en memoria. La división se lleva a cabo con una función de asociación en los atributos de la reunión, de tal modo que los pares de particiones correspondientes se puedan reunir independientemente.
- La eliminación de duplicados, la proyección, las operaciones de conjuntos (unión, intersección y diferencia) se pueden realizar mediante ordenación o asociación.
- Las operaciones de reunión externa se pueden implementar como extensiones simples de los algoritmos de reunión.
- Las técnicas de asociación y ordenación son duales, en el sentido en que muchas operaciones como la eliminación de duplicados, agregación, reuniones y reuniones externas se pueden implementar mediante asociación o bien por ordenación.
- Una expresión se puede evaluar mediante materialización, donde el sistema calcula el resultado de cada subexpresión y lo almacena en disco, y después lo usa para calcular el resultado de la expresión padre.
- El encauzamiento ayuda a evitar la escritura en disco de los resultados de muchas subexpresiones, usando los resultados de la expresión padre como si se estuviesen generando.

TÉRMINOS DE REPASO

- Árbol de operadores
- Búsqueda binaria
- Búsqueda lineal
- Ciclos
- E/S paralela
- E/S secuencial
- Equirreunión
- Evaluación encauzada
  - Cauce bajo demanda (perezoso, extracción)
  - Cauce por productor (impaciente, inserción)
  - Iterador
- Evaluación materializada
- Evaluación de primitivas
- Evitación del desbordamiento
  - Factor de escape
  - Probar
  - Resolución del desbordamiento
  - Sesgo
- Exploración de archivos
- Exploración de índices
- Índice compuesto
- Intersección de identificadores

- Mediadas del coste de las consultas
- Memoria intermedia doble
- Mezcla de n vías
- Motor de ejecución de consultas
- Ordenación externa
- Ordenación-mezcla externa
- Plan de ejecución de consultas
- Plan de evaluación de consultas
- Procesamiento de consultas
- Reunión por asociación
  - Construir
  - Desbordamiento de la tabla asociativa
  - División recursiva
  - Entrada para construir
  - Entrada para probar
- Reunión por asociación híbrida
- Reunión en bucle anidado
- Reunión en bucle anidado por bloques
- Reunión en bucle anidado indexada
- Reunión encauzada
- Reunión por mezcla
- Reunión por mezcla híbrida
- Reunión por ordenación-mezcla
- Rutas de acceso
- Selección conjuntiva
- Selección disyuntiva
- Selecciones usando índices

EJERCICIOS

13.1. ¿Por qué no hay que obligar a los usuarios a que elijan explícitamente una estrategia de procesamiento de la consulta? ¿Hay casos en los que *es* deseable que los usuarios sepan el coste de las distintas estrategias posibles? Razónese la respuesta.

13.2. Considérese la siguiente consulta SQL para la base de datos bancaria:

select *T.nombre-sucursal*
from *sucursal T, sucursal S*
where *T.activo > S.activo* **and** *S.ciudad-sucursal* = «Arganzuela»

Escríbase una expresión del álgebra relacional equivalente a la dada que sea más eficiente. Justifíquese la elección.

13.3. ¿Cuáles son las ventajas e inconvenientes de los índices asociativos en relación con índices de árbol B$^+$? ¿Cómo podría el tipo de índice influenciar en la elección de una estrategia de procesamiento de una consulta?

13.4. Supóngase (para simplificar este ejercicio) que solamente cabe una tupla en un bloque y que la memoria puede contener como máximo tres marcos de página. Muéstrense las secuencias creadas en cada ciclo del algoritmo de ordenación-mezcla cuando se aplica para ordenar el primer atributo de las siguientes tuplas: (canguro, 17), (ualabí, 21), (emú, 1), (wombat, 13), (ornitorrinco, 3), (león, 8), (jabalí, 4), (cebra, 11), (koala, 6), (hiena, 9), (cálao, 2), (babuino, 12).

13.5. Dadas las relaciones $r_1(A, B, C)$ y $r_2(C, D, E)$ con las siguientes propiedades: r_1 tiene 20.000 tuplas, r_2 tiene 45.000 tuplas, 25 tuplas de r_1 caben en un bloque y 30 tuplas de r_2 que caben en un bloque. Estímese el número de accesos a bloques requeridos utilizando las siguientes estrategias para la reunión $r_1 \bowtie r_2$:

a. Reunión en bucle anidado
b. Reunión en bucle anidado por bloques
c. Reunión por mezcla
d. Reunión por asociación

13.6. Diséñese una variante del algoritmo híbrido de reunión por mezcla para el caso en el que las dos relaciones no están ordenadas según el orden físico de almacenamiento, pero ambas tienen un índice secundario ordenado en los atributos de la reunión.

13.7. El algoritmo de reunión en bucle anidado indexada descrito en el Apartado 13.5.3 puede ser ineficiente si el índice fuera secundario y hubiese varias tuplas con el mismo valor en los atributos de la reunión. ¿Por qué es ineficiente? Descríbase una forma de reducir el coste de recuperar las tuplas de la relación más interna utilizando ordenación. ¿Bajo qué condiciones sería este algoritmo más eficiente que la reunión por mezcla híbrida?

13.8. Estímese el número de accesos a bloques necesitados por la solución del Ejercicio 13.6 para $r_1 \bowtie r_2$, donde r_1 y r_2 son como las relaciones definidas en el Ejercicio 13.5.

13.9. Sean *r* y *s* dos relaciones sin índices que no están ordenadas. Suponiendo una memoria infinita ¿cuál es la manera más económica (en términos de operaciones de E/S) para calcular $r \bowtie s$? ¿Cuánta memoria se necesita en este algoritmo?

13.10. Supóngase que hay un índice de árbol B$^+$ disponible en *ciudad-sucursal* de la relación *sucursal* y que no

hay más índices. ¿Cuál sería el mejor modo de manejar las siguientes selecciones con negaciones?

a. $\sigma_{\neg\,(ciudad\text{-}sucursal\,<\,\text{«Arganzuela»})}\,(sucursal)$
b. $\sigma_{\neg\,(ciudad\text{-}sucursal\,=\,\text{«Arganzuela»})}\,(sucursal)$
c. $\sigma_{\neg\,(ciudad\text{-}sucursal\,<\,\text{«Arganzuela»}\,\vee\,activo\,<\,5000)}\,(sucursal)$

13.11. El algoritmo de reunión por asociación descrito en el Apartado 13.5.5 calcula la reunión natural de dos relaciones. Descríbase cómo extender el algoritmo de reunión por asociación para calcular la reunión externa por la izquierda, la reunión externa por la derecha y la reunión externa completa. (Sugerencia: se puede mantener información adicional con cada tupla en el índice asociativo para detectar si alguna tupla en la relación para probar concuerda con alguna tupla del índice asociativo.) Compruébese el algoritmo con las relaciones *cliente* e *impositor*.

13.12. Escríbase el pseudocódigo para un iterador que implemente la reunión en bucle anidado indexada, donde la relación externa esté encauzada. Utilícense las funciones de iterador estándares en el pseudocódigo. Muéstrese el estado del iterador entre las llamadas.

13.13. Diséñense algoritmos basado en ordenación y asociación para el cálculo de la operación división.

NOTAS BIBLIOGRÁFICAS

Todos los procesadores de consultas deben analizar instrucciones del lenguaje de consulta y deben traducirlas a su formato interno. El análisis de los lenguajes de consulta difiere poco del análisis de los lenguajes de programación tradicionales. La mayoría de los libros sobre compiladores, como Aho et al. [1986], tratan las principales técnicas de análisis y presentan la optimización desde el punto de vista de los lenguajes de programación.

Knuth [1973] presenta un excelente descripción de algoritmos de ordenación externa, incluyendo una optimización que puede originar secuencias iniciales que son (en media) el doble del tamaño de la memoria. Basados en estudios del rendimiento realizados a mediados de 1970, los sistemas de bases de datos de esa época utilizaban solamente reunión en bucle anidado y reunión por mezcla. Estos estudios, que estuvieron relacionados con el desarrollo de System R, determinaron que tanto la reunión en bucle anidado como la reunión por mezcla casi siempre proporcionaban el método de reunión óptimo [Blasgen y Eswaran 1976]; por tanto, estos son los dos únicos algoritmos de reunión implementados en System R. Sin embargo, el estudio de System R no incluyó el análisis de los algoritmos de reunión por asociación. Actualmente, estos algoritmos se consideran muy eficientes.

Los algoritmos de reunión por asociación se desarrollaron inicialmente para sistemas de bases de datos paralelos. La técnica de reunión por asociación se describe en Kitsuregawa et al. [1983] y en Shapiro [1986] se describen extensiones incluyendo la reunión por asociación híbrida. Resultados más recientes de Zeller y Gray [1990] y de Davison y Graefe [1994] describen técnicas de reunión por asociación que se pueden adaptar a la memoria disponible, que es importante en sistemas donde se pueden ejecutar a la vez varias consultas. Graefe et al. [1998] describe el uso en Microsoft SQL Server de las reuniones por asociación y los *equipos de asociación*, que permiten el encauzamiento de las reuniones por asociación usando la misma división para todas las reunión en una secuencia encauzada.

Graefe [1993] presenta una excelente revisión de las técnicas de evaluación de consultas. Una revisión anterior de las técnicas de procesamiento de consultas aparece en Jarke y Koch [1984].

El procesamiento de consultas en bases de datos en memoria principal se trata en DeWitt et al. [1984] y Whang y Krishnamurthy [1990]. Kim [1982] y Kim [1984] describen estrategias de reunión y el uso óptimo de la memoria principal disponible.

CAPÍTULO

14 OPTIMIZACIÓN DE CONSULTAS

La **optimización de consultas** es el proceso de selección del plan de evaluación de las consultas más eficiente de entre las muchas estrategias generalmente disponibles para el procesamiento de una consulta dada, especialmente si la consulta es compleja. No se espera que los usuarios escriban las consultas de modo que puedan procesarse de manera eficiente. Por el contrario, se espera que el sistema cree un plan de evaluación de las consultas que minimice el coste de la evaluación de las consultas. Aquí es donde entra en acción la optimización de consultas.

Un aspecto de la optimización de las consultas tiene lugar en el nivel del álgebra relacional, donde el sistema intenta hallar una expresión que sea equivalente a la expresión dada, pero de ejecución más eficiente. Otro aspecto es la elección de una estrategia detallada para el procesamiento de la consulta, como puede ser la selección del algoritmo que se utilizará para ejecutar una operación, la selección de los índices concretos que se van a emplear, etcétera.

La diferencia en coste (en términos de tiempo de evaluación) entre una estrategia buena y una mala suele ser sustancial, y puede ser de varios órdenes de magnitud. Por tanto, merece la pena que el sistema pase una cantidad importante de tiempo en la selección de una buena estrategia para el procesamiento de la consulta, aunque esa consulta sólo se ejecute una vez.

14.1. VISIÓN GENERAL

Considérese la expresión del álgebra relacional para la consulta «Hallar los nombres de todos los clientes que tengan una cuenta en cualquier sucursal ubicada en Arganzuela».

$$\Pi_{\textit{nombre-cliente}} (\sigma_{\textit{ciudad-sucursal} = \text{«Arganzuela»}} (\textit{sucursal} \bowtie (\textit{cuenta} \bowtie \textit{impositor})))$$

Esta expresión crea una relación intermedia de gran tamaño, *sucursal* ⋈ *cuenta* ⋈ *impositor*. Sin embargo, sólo nos interesan unas pocas tuplas de esta relación (las correspondientes a las sucursales ubicadas en Arganzuela), y sólo en uno de los seis atributos de la relación. Dado que sólo nos preocupan las tuplas de la relación *sucursal* que corresponden a las sucursales ubicadas en Arganzuela, no hace falta considerar las tuplas que no tienen *ciudad-sucursal* = «Arganzuela». Al reducir el número de tuplas de la relación *sucursal* a las que hace falta tener acceso, se reduce el tamaño del resultado intermedio. La consulta queda ahora representada por la expresión de álgebra relacional:

$$\Pi_{\textit{nombre-cliente}} (\sigma_{\textit{ciudad-sucursal} = \text{«Arganzuela»}} (\textit{sucursal})) \bowtie (\textit{cuenta} \bowtie \textit{impositor}))$$

que es equivalente a la expresión algebraica original, pero que genera relaciones intermedias de menor tamaño. La Figura 14.1 muestra la expresión inicial y la transformada.

Dada una expresión del álgebra relacional, es labor del optimizador de consultas diseñar un plan de evaluación de consultas que calcule el mismo resultado que la expresión dada, y que sea la manera menos costosa de generar ese resultado (o, como mínimo, que no sea mucho más costoso que la manera menos costosa).

Para escoger entre los diferentes planes de evaluación de consultas el optimizador tiene que **estimar** el coste de cada plan de evaluación. El cálculo del coste

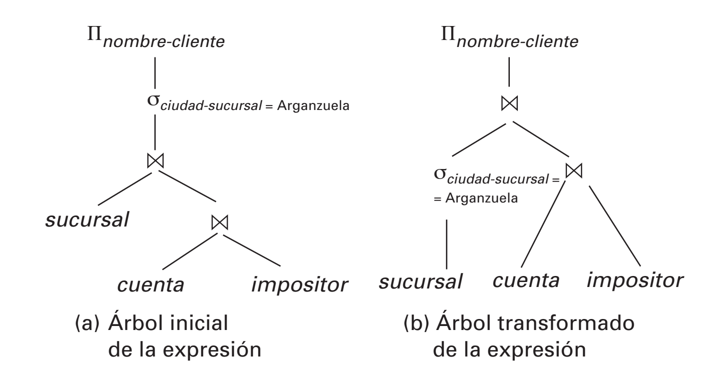


FIGURA 14.1. Expresiones equivalentes.

exacto de evaluación de un plan no suele resultar posible sin evaluar realmente el plan. En lugar de eso, los optimizadores hacen uso de la información estadística sobre las relaciones, como los tamaños de las relaciones y las profundidades de los índices, para realizar una buena estimación del coste de cada plan. El acceso a los discos, que resulta lento en comparación con el acceso a la memoria, suele dominar el coste del procesamiento de las consultas.

En el Apartado 14.2 se describe el modo de estimar las estadísticas de los resultados de cada operación en los planes de consultas. El empleo de estas estadísticas con las fórmulas de costes del Capítulo 13 permite estimar los costes de cada operación. Los costes individuales se combinan para determinar el coste estimado de la evaluación de la expresión de álgebra relacional dada, como se indicó previamente en el Apartado 13.7.

Para hallar el plan de evaluación de consultas menos costoso el optimizador necesita generar planes alternativos que produzcan el mismo resultado que la expresión dada y escoger el menos costoso. La generación de planes de evaluación de consultas implica dos etapas: (1) la generación de expresiones que sean equivalentes lógicamente a la expresión dada y (2) la anotación de las expresiones resultantes en maneras alternativas de generar planes de evaluación de consultas alternativos. Las dos etapas están entrelazadas en el optimizador de consultas –se generan y se anotan unas expresiones, luego se generan y se anotan otras expresiones, etcétera.

Para implementar la primera etapa el optimizador de consultas debe generar expresiones equivalentes a la expresión dada. Lo hace mediante las *reglas de equivalencia,* que especifican el modo de transformar una expresión en otra equivalente lógicamente. Estas reglas se describen en el Apartado 14.3.1. En el Apartado 14.4 se describe el modo de escoger un plan de evaluación de consultas. Se puede escoger uno basado en el coste estimado de los planes. Dado que el coste es una estimación, el plan seleccionado no es necesariamente el menos costoso; no obstante, siempre y cuando las estimaciones sean buenas, es probable que el plan sea el menos costoso, o no mucho más costoso. Esta optimización, denominada **optimización basada en costes**, se describe en el Apartado 14.4.2.

Las vistas materializadas ayudan a acelerar el procesamiento de ciertas consultas. En el Apartado 14.5 se estudia el modo de «mantener» las vistas materializadas –es decir, mantenerlas actualizadas– y la manera de llevar a cabo la optimización de consultas con las vistas materializadas.

14.2. ESTIMACIÓN DE LAS ESTADÍSTICAS DE LOS RESULTADOS DE LAS EXPRESIONES

El coste de cada operación depende del tamaño y de otras estadísticas de sus valores de entrada. Dada una expresión como $a \bowtie (b \bowtie c)$, para estimar el coste de combinar a con $(b \bowtie c)$ hay que hacer estimaciones de estadísticas como el tamaño de $b \bowtie c$.

En este apartado se relacionarán en primer lugar algunas estadísticas de las relaciones de bases de datos que se almacenan en los catálogos de los sistemas de bases de datos y luego se mostrará el modo de utilizar las estadísticas para estimar estadísticas de los resultados de varias operaciones relacionales.

Una cosa que quedará clara más adelante en este apartado es que las estimaciones no son muy precisas, ya que se basan en suposiciones que puede que no se cumplan exactamente. El plan de evaluación de consultas que tenga el coste estimado de ejecución más reducido puede, por tanto, no tener el coste real de ejecución más bajo. Sin embargo, la experiencia real ha mostrado que, aunque las estimaciones no sean muy precisas, los planes con los costes estimados más reducidos tienen costes de ejecución reales que son los más reducidos o se hallan cercanos a los costes reales de ejecución más bajos.

14.2.1. Información del catálogo

Los catálogos de los SGDD almacenan la siguiente información estadística sobre las relaciones de las bases de datos:

- n_r, el número de tuplas de la relación r.
- b_r, el número de bloques que contienen tuplas de la relación r.
- t_r, el tamaño de cada tupla de la relación r en bytes.
- f_r, el factor de bloqueo de la relación r, es decir, el número de tuplas de la relación r que caben en un bloque.
- $V(A, r)$, el número de valores distintos que aparecen en la relación r para el atributo A. Este valor es igual que el tamaño de $\Pi_A(r)$. Si A es una clave de la relación r, $V(A, r)$ es n_r.

La última estadística, $V(A, r)$, también puede calcularse para conjuntos de atributos, si se desea, en vez de sólo para atributos aislados. Por tanto, dado un conjunto de atributos, A, $V(A, r)$ es el tamaño de $\Pi_A(r)$.

Si se supone que las tuplas de la relación *r* se almacenan físicamente juntas en un archivo, se cumple la ecuación siguiente:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

Las estadísticas sobre los índices, como las alturas de los árboles B$^+$ y el número de páginas hojas de los índices, también se conservan en el catálogo.

Si se desean conservar estadísticas precisas, por tanto, cada vez que se modifica una relación también hay que actualizar las estadísticas. Esta actualización supone una sobrecarga sustancial. Por tanto, la mayor parte de los sistemas no actualizan las estadísticas con cada modificación. En lugar de eso, actualizan las estadísticas durante los periodos de poca carga del sistema. En consecuencia, puede que las estadísticas utilizadas para escoger una estrategia de procesamiento de consultas no sean completamente exactas. Sin embargo, si no se producen demasiadas actualizaciones en los intervalos entre las actualizaciones de las estadísticas, éstas serán lo bastante precisas como para proporcionar una buena estimación de los costes relativos de los diferentes planes.

La información estadística anotada aquí está simplificada. Los optimizadores verdaderos suelen conservar información estadística adicional para mejorar la precisión de sus estimaciones de costes de los planes de evaluación. Por ejemplo, algunas bases de datos almacenan la distribución de los valores de cada atributo en forma de **histograma**: en los histogramas los valores del atributo se dividen en una serie de rangos, y con cada rango el histograma asocia el número de tuplas cuyo valor del atributo se halla en ese rango. Como ejemplo de histograma, el rango de valores del atributo *edad* de la relación *persona* puede dividirse en 0-9, 10-19, . . . , 90-99 (suponiendo una edad máxima de 99). Con cada rango se almacena un recuento del número de tuplas *persona* cuyos valores de *edad* se hallan en ese rango. Sin la información del histograma un optimizador tendría que suponer que la distribución de los valores es uniforme; es decir, que cada rango tiene el mismo recuento.

14.2.2. Estimación del tamaño de la selección

La estimación del tamaño del resultado de una operación de selección depende del predicado de la selección. En primer lugar se considerará un solo predicado de igualdad, luego un solo predicado de comparación y, finalmente, combinaciones de predicados.

- $\sigma_{A=a}(r)$: Si se supone una distribución uniforme de los valores (es decir, que cada valor aparece con igual probabilidad), se puede estimar que el resultado de la selección tiene $n_r / V(A, r)$ tuplas, suponiendo que el valor *a* aparece en el atributo *A* de algún registro de *r*. La suposición de que el valor *a* de la selección aparece en algún registro suele ser cierta, y las estimaciones de costes suelen hacerla de manera implícita. No obstante, no suele ser realista suponer que cada valor aparece con igual probabilidad. El atributo *nombre-sucursal* de la relación *cuenta* es un ejemplo en el que esta suposición no es válida. Hay una tupla de la relación *cuenta* para cada cuenta. Resulta razonable esperar que las sucursales grandes tengan más cuentas que las pequeñas. Por tanto, algunos valores *nombre-sucursal* aparecen con mayor probabilidad que otros. Pese al hecho de que la suposición de distribución uniforme no suele ser correcta, resulta una aproximación razonable de la realidad en muchos casos, y ayuda a mantener la representación relativamente sencilla.
- $\sigma_{A \le v}(r)$: Considérese una selección de la forma $\sigma_{A \le v}(r)$. Si el valor real utilizado en la comparación (v) está disponible en el momento de la estimación del coste, puede hacerse una estimación más precisa. Los valores mínimo y máximo (min (A, r) y max (A, r)) del atributo pueden almacenarse en el catálogo. Suponiendo que los valores están distribuidos de manera uniforme, se puede estimar el número de registros que cumplirán la condición $A \le v$ como 0 si $v <$ min (A, r), como n_r si $v \ge$ max (A, r) y como

  $$n_r \cdot \frac{v - \min(A, r)}{\max(A, r) - \min(A, r)}$$

  en otro caso.

  En algunos casos, como cuando la consulta forma parte de un procedimiento almacenado, puede que el valor v no esté disponible cuando se optimice la consulta. En esos casos, se supondrá que aproximadamente la mitad de los registros cumplen la condición de comparación. Es decir, se supone que el resultado tiene $n_r/2$ tuplas; la estimación puede resultar muy imprecisa, pero es lo mejor que se puede hacer sin más información.
- Selecciones complejas:
  - **Conjunción:** Una *selección conjuntiva* es una selección de la forma

    $$\sigma_{\theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_n}(r)$$

    Se puede estimar el tamaño del resultado de esta selección: Para cada θ_i, se estima el tamaño de la selección $\sigma_{\theta_i}(r)$, denotada por s_i, como se ha descrito anteriormente. Por tanto, la probabilidad de que una tupla de la relación satisfaga la condición de selección θ_i es s_i/n_r.

    La probabilidad anterior se denomina **selectividad** de la selección σ_{θ_i} (r). Suponiendo que las condiciones sean *independientes* entre sí, la probabilidad de que una tupla satisfaga todas las condiciones es simplemente el producto de todas

estas probabilidades. Por tanto, se estima el número de tuplas de la selección completa como

$$n_r * \frac{s_1 * s_2 * \cdots * s_n}{n_r^n}$$

– **Disyunción:** Una *selección disyuntiva* es una selección de la forma

$$\sigma_{\theta_1 \vee \theta_2 \vee \ldots \vee \theta_n}(r)$$

Una condición disyuntiva se satisface por la unión de todos los registros que satisfacen las condiciones individuales y simples θ_i.

Como anteriormente, admitamos que s_i / n_r denota la probabilidad de que una tupla satisfaga la condición θ_i. La probabilidad de que la tupla satisfaga la disyunción es, pues, 1 menos la probabilidad de que no satisfaga *ninguna* de las condiciones:

$$1 - (1 - \frac{s_1}{n_r}) * (1 - \frac{s_2}{n_r}) * \cdots * (1 - \frac{s_n}{n_r})$$

Multiplicando este valor por n_r se obtiene el número estimado de tuplas que satisfacen la selección.

– **Negación:** En ausencia de valores nulos el resultado de una selección $\sigma_\theta(r)$ es simplemente las tuplas de r que no están en $\sigma_\theta(r)$. Ya se sabe el modo de estimar el número de tuplas de $\sigma_\theta(r)$. El número de tuplas de $\sigma_\theta(r)$ se estima, por lo tanto, que es $n(r)$ menos el número estimado de tuplas de $\sigma_\theta(r)$.

Se pueden tener en cuenta los valores nulos estimando el número de tuplas para las que la condición θ se evalúa como *desconocida*, y restar ese número de la estimación anterior que ignora los valores nulos. La estimación de ese número exige conservar estadísticas adicionales en el catálogo.

14.2.3. Estimación del tamaño de las reuniones

En este apartado se verá el modo de estimar el tamaño del resultado de una reunión.

El producto cartesiano $r \times s$ contiene $n_r * n_s$ tuplas. Cada tupla de $r \times s$ ocupa $t_r + t_s$ bytes, de donde se puede calcular el tamaño del producto cartesiano.

La estimación del tamaño de una reunión natural resulta algo más complicada que la estimación del tamaño de una selección del producto cartesiano. Sean $r(R)$ y $s(S)$ dos relaciones.

- Si $R \cap S = \emptyset$ —es decir, las relaciones no tienen ningún atributo en común— entonces $r \bowtie s$ es igual que $r \cap s$, y se puede utilizar la técnica de estimación anterior para los productos cartesianos.
- Si $R \cap S$ es una clave de R, entonces se sabe que cada tupla de s se combinará como máximo con una tupla de r. Por tanto, el número de tuplas de $r \bowtie s$ no es mayor que el número de tuplas de s. El caso de que $R \cap S$ sea una clave de S es simétrico al caso que se acaba de describir. Si $R \cap S$ forma una clave externa de S, que haga referencia a R, el número de tuplas de $r \bowtie s$ es exactamente el mismo que el número de tuplas de s.
- El caso más difícil es que $R \cap S$ no sea una clave de R ni de S. En este caso se supone, como se hizo para las selecciones, que todos los valores aparecen con igual probabilidad. Considérese una tupla t de r y supóngase que $R \cap S = \{A\}$. Se estima que la tupla t produce

$$\frac{n_s}{V(A, s)}$$

tuplas en $r \bowtie s$, ya que este número es el número promedio de tuplas de s con un valor dado para los atributos A. Considerando todas las tuplas de r se estima que hay

$$\frac{n_r * n_s}{V(A, s)}$$

tuplas in $r \bowtie s$. Obsérvese que, si se invierten los papeles de r y de s en la estimación anterior, se obtiene una estimación de

$$\frac{n_r * n_s}{V(A, r)}$$

tuplas en $r \bowtie s$. Estas dos estimaciones son diferentes si $V(A, r) \neq V(A, s)$. Si esta situación se produce, probablemente haya tuplas pendientes que no participen en la reunión. Por tanto, probablemente la menor de la dos estimaciones sea la más precisa.

La estimación anterior del tamaño de la reunión puede ser demasiado elevada si los valores de $V(A, r)$ para el atributo A en r tienen pocas tuplas en común con los valores de $V(A, s)$ para el atributo A en s. No obstante, es improbable que se dé esta situación en la realidad, ya que las tuplas pendientes o bien no existen o sólo constituyen una pequeña fracción de las tuplas, en la mayor parte de las relaciones reales. Lo que es más importante todavía, la estimación anterior depende de la suposición de que todos los valores aparecen con igual probabilidad. Hay que utilizar técnicas más sofisticadas para la estimación del tamaño si esta suposición no resulta válida.

Se puede estimar el tamaño de una reunión theta $r \bowtie_\theta s$ reescribiendo la reunión como $\sigma_\theta(r \times s)$ y empleando las estimaciones de tamaño de los productos cartesianos junto con las estimaciones de tamaño de las selecciones, que se vieron en el Apartado 14.2.2.

Para ilustrar todas estas maneras de estimar el tamaño de las reuniones, considérese la expresión

impositor ⋈ *cliente*

Supóngase que se dispone de la siguiente información de catálogo sobre las dos relaciones:

- $n_{cliente}$ = 10000.
- $f_{cliente}$ = 25, lo que implica que $b_{cliente}$ = = 10000/25 = 400.
- $n_{impositor}$ = 5000.
- $f_{impositor}$ = 50, lo que implica que $b_{impositor}$ = = 5000/50 = 100.
- *V* (nombre-cliente, impositor) = 2.500, lo que implica que, en promedio, cada cliente tiene dos cuentas.

Supóngase también que *nombre-cliente* de *impositor* es una clave externa de *cliente*.

En el ejemplo de *impositor* ⋈ *cliente*, *nombre-cliente* de *impositor* es una clave externa que hace referencia a *cliente*; por tanto, el tamaño del resultado es exactamente $n_{impositor}$, que es 5000.

Calculemos ahora las estimaciones de tamaño de *impositor* ⋈ *cliente* sin utilizar la información sobre las claves externas. Como *V* (*nombre-cliente, impositor*) = 2500 y *V* (*nombre-cliente, cliente*) = 10000, las dos estimaciones que se consiguen son 5000 ∗ 10000/2500 = 20000 y 5000 ∗ 10000/10000 = 5000, y se escoge la menor. En este caso, la menor de las estimaciones es igual que la que se calculó anteriormente a partir de la información sobre las claves externas.

14.2.4. Estimación del tamaño de otras operaciones

A continuación se esbozará el modo de estimar el tamaño de los resultados de otras operaciones del álgebra relacional.

Proyección: El tamaño estimado (número de registros de las tuplas) de una proyección de la forma $\prod_A (r)$ es *V* (*A, r*), ya que la proyección elimina los duplicados.

Agregación: El tamaño de $_A \mathcal{G}_F (r)$ es simplemente *V* (*A, r*), ya que hay una tupla de $_A \mathcal{G}_F (r)$ por cada valor distinto de *A*.

Operaciones de conjuntos: Si las dos entradas de una operación de conjuntos son selecciones de la misma relación se puede reescribir la operación de conjuntos como disyunciones, conjunciones o negaciones. Por ejemplo, $\sigma_{\theta_1}(r) \cup \sigma_{\theta_2}(r)$ puede reescribirse como $\sigma_{\theta_1 \vee \theta_2}(r)$. De manera parecida, se pueden reescribir las intersecciones como conjunciones y la diferencia de conjuntos empleando la negación, siempre que las dos relaciones que participan en la operación de conjuntos sean selecciones de la misma relación. Luego se pueden utilizar las estimaciones de las selecciones que impliquen conjunciones, disyunciones y negaciones del Apartado 14.2.2.

Si las entradas no son selecciones de la misma relación se estiman los tamaños de esta manera: El tamaño estimado de $r \cup s$ es la suma de los tamaños de *r* y de *s*. El tamaño estimado de $r \cap s$ es el mínimo de los tamaños de *r* y de *s*. El tamaño estimado de $r - s$ es el mismo tamaño de *r*. Las tres estimaciones pueden ser imprecisas, pero proporcionan cotas superiores para los tamaños.

Reunión externa: El tamaño estimado de *r* ⟕ *s* es el tamaño de *r* ⋈ *s* más el tamaño de *r*; el de *r* ⟖ *s* es simétrico, mientras que el de *r* ⟗ *s* es el tamaño de *r* ⋈ *s* más los tamaños de *r* y de *s*. Las tres estimaciones pueden ser imprecisas, pero proporcionan cotas superiores para los tamaños.

14.2.5. Estimación del número de valores distintos

Para las selecciones el número de valores distintos de un atributo (o de un conjunto de atributos) *A* en el resultado de una selección, $V(A, \sigma_\theta(r))$, puede estimarse de las maneras siguientes:

- Si la condición de selección θ obliga a que *A* adopte un valor especificado (por ejemplo, *A* = 3), $V(A, \sigma_\theta(r)) = 1$.
- Si θ obliga a que *A* adopte un valor de entre un conjunto especificado de valores (por ejemplo, $(A = 1 \vee A = 3 \vee A = 4)$), entonces $V(A, \sigma_\theta(r))$ se define como el número de valores especificados.
- Si la condición de selección θ es de la forma *A op v*, donde *op* es un operador de comparación, $V(A, \sigma_\theta(r))$ se estima que es $V(A, r) * s$, donde *s* es la selectividad de la selección.
- En todos los demás casos de selecciones se da por supuesto que la distribución de los valores de *A* es independiente de la distribución de los valores para los que se especifican las condiciones de selección y se utiliza una estimación aproximada de min $(V(A, r), n_{\sigma_\theta(r)})$. Se puede obtener una estimación más precisa para este caso utilizando la teoría de la probabilidad, pero la aproximación anterior funciona bastante bien.

Para las reuniones el número de valores distintos de un atributo (o de un conjunto de atributos) *A* en el resultado de una reunión, *V* (*A, r* ⋈ *s*), puede estimarse de las maneras siguientes:

- Si todos los atributos de *A* proceden de *r*, *V* (*A, r* ⋈ *s*) se estima como min $(V(A, r), n_{r \bowtie s})$, y de manera parecida si todos los atributos de *A* proceden de> *s*, *V* (*A, r* ⋈ *s*) se estima que es min $(V(A, s), n_{r \bowtie s})$.

- Si A contiene atributos $A1$ de r y $A2$ de s, entonces $V(A, r \bowtie s)$ se estima como min $(V(A1, r) * V(A2 - A1, s), V(A1 - A2, r) * V(A2, s), n_{r \bowtie s})$ Obsérvese que algunos atributos pueden estar en $A1$ y en $A2$, y que $A1$-$A2$ y $A2$-$A1$ denotan, respectivamente, a los atributos de A que sólo proceden de r y a los atributos de A que sólo proceden de s. Nuevamente, se pueden obtener estimaciones más precisas utilizando la teoría de la probabilidad, pero las aproximaciones anteriores funcionan bastante bien.

Las estimaciones de los distintos valores son directas para las proyecciones: son iguales en $\Pi_A(r)$ que en r. Lo mismo resulta válido para los atributos de agrupación de las agregaciones. Para los resultados de **suma**, **cuenta** y **promedio**, se puede suponer, por sencillez, que todos los valores agregados son distintos. Para **min** (A) y **max** (A), el número de valores distintos puede estimarse como min $(V(A, r), V(G, r))$, donde G denota los atributos de agrupamiento. Se omiten los detalles de la estimación de los valores distintos para otras operaciones.

14.3. TRANSFORMACIÓN DE EXPRESIONES RELACIONALES

Hasta ahora se han estudiado los algoritmos para evaluar las operaciones extendidas del álgebra relacional y hacer estimaciones de sus costes. Como se mencionó al comienzo de este capítulo, las consultas se pueden expresar de varias maneras diferentes, con costes de evaluación diferentes. En este apartado, en lugar de tomar la expresión relacional como se da, se consideran expresiones alternativas equivalentes.

Se dice que dos expresiones del álgebra relacional son **equivalentes** si, en cada ejemplar legal de la base de datos, las dos expresiones generan el mismo conjunto de tuplas. (Recuérdese que un ejemplar legal de la base de datos es la que satisface todas las restricciones de integridad especificadas en el esquema de la base de datos.) Obsérvese que el orden de las tuplas resulta irrelevante; puede que las dos expresiones generen las tuplas en órdenes diferentes, pero se considerarán equivalentes siempre que el conjunto de tuplas sea el mismo.

En SQL las entradas y las salidas son multiconjuntos de tuplas, y se utiliza la versión para multiconjuntos del álgebra relacional para evaluar las consultas de SQL. Se dice que dos expresiones de la versión *para multiconjuntos* del álgebra relacional son equivalentes si en cada base de datos legal las dos expresiones generan el mismo multiconjunto de tuplas. El estudio de este capítulo se basa en el álgebra relacional. Las extensiones a la versión para multiconjuntos del álgebra relacional se dejan al lector como ejercicios.

14.3.1. Reglas de equivalencia

Una **regla de equivalencia** dice que las expresiones de dos formas son equivalentes. Se puede sustituir una expresión de la primera forma por una expresión de la segunda forma, o viceversa —es decir, se puede sustituir una expresión de la segunda forma por una expresión de la primera forma—, ya que las dos expresiones generan el mismo resultado en cualquier base de datos válida. El optimizador utiliza las reglas de equivalencia para transformar las expresiones en otras equivalentes lógicamente.

A continuación se relacionan varias reglas generales de equivalencia para las expresiones del álgebra relacional. Algunas de las equivalencias relacionadas aparecen en la Figura 14.2. Se utilizan θ, θ_1, θ_2, etcétera, para denotar los predicados, L_1, L_2, L_3, etcétera, para denotar las listas de atributos y E, E_1,E_2, etcétera, para denotar las expresiones del álgebra relacional. El nombre de relación r no es más que un caso especial de expresión del álgebra relacional y puede utilizarse siempre que aparezca E.

1. Las operaciones de selección conjuntivas pueden dividirse en una secuencia de selecciones individuales. Esta transformación se denomina cascada de σ.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

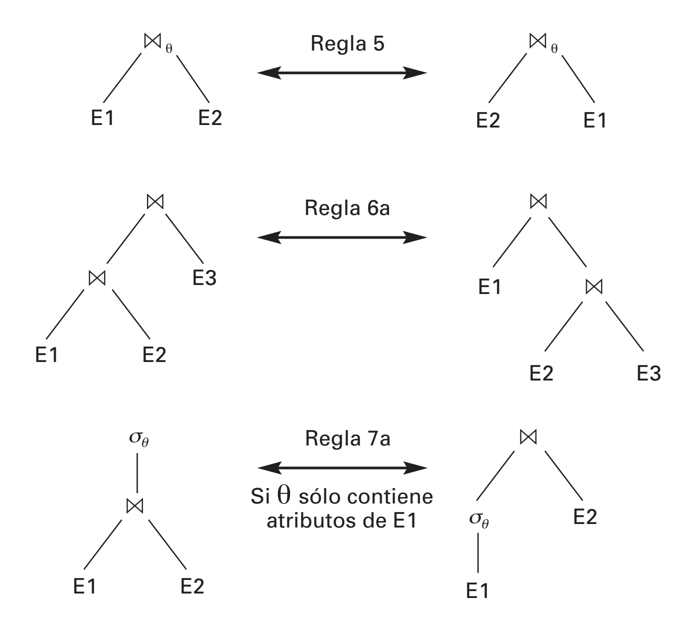


FIGURA 14.2. Representación gráfica de las equivalencias.

2. Las operaciones de selección son **conmutativas**.

$$\sigma_{\theta_1} (\sigma_{\theta_2} (E)) = \sigma_{\theta_2} (\sigma_{\theta_1} (E))$$

3. Sólo son necesarias las últimas operaciones de una secuencia de operaciones de proyección, las demás pueden omitirse. Esta transformación también puede denominarse cascada de Π.

$$\Pi_{L_1} (\Pi_{L_2} (\ldots (\Pi_{L_1} (E)) \ldots)) = \Pi_{L_1} (E)$$

4. Las selecciones pueden combinarse con los productos cartesianos y con las reuniones zeta.

   a. $\sigma_\theta (E_1 \times E_2) = E_1 \bowtie_\theta E_2$
   Esta expresión es precisamente la definición de la reunión zeta.

   b. $\sigma_{\theta_1} (E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

5. Las operaciones de reunión zeta son conmutativas.

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

Realmente, el orden de los atributos es diferente en el término de la derecha y en el de la izquierda, por lo que la equivalencia no se cumple si se tiene en cuenta el orden de los atributos. Se puede añadir una operación de proyección a uno de los lados de la equivalencia para reordenar los atributos de la manera adecuada, pero por simplicidad se omite la proyección y se ignora el orden de los atributos en la mayor parte de los ejemplos.

Recuérdese que el operador de reunión natural es simplemente un caso especial del operador de reunión zeta; por tanto, las reuniones naturales también son conmutativas.

6. **a.** Las operaciones de reunión natural son **asociativas**.

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

   b. Las reuniones zeta son asociativas en el sentido siguiente:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 =$$
$$= E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

   donde θ_2 implica solamente atributos de E_2 y de E_3. Cualquiera de estas condiciones puede estar vacía; por tanto, se deduce que la operación producto cartesiano ($\times$) también es asociativa. La conmutatividad y la asociatividad de las operaciones de reunión son importantes para la reordenación de las reuniones en la optimización de las consultas.

7. La operación de selección se distribuye por la operación de reunión zeta bajo las dos condiciones siguientes:

   a. Se distribuye cuando todos los atributos de la condición de selección θ_0 implican únicamente los atributos de una de las expresiones (por ejemplo, E_1) que se están reuniendo.

$$\sigma_{\theta_0} (E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0} (E_1)) \bowtie_\theta E_2$$

   b. Se distribuye cuando la condición de selección θ_1 implica únicamente los atributos de E_1 y θ_2 implica únicamente los atributos de E_2.

$$\sigma_{\theta_1 \wedge \theta_2} (E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1} (E_1)) \bowtie_\theta (\sigma_{\theta_2} (E_2))$$

8. La operación proyección se distribuye por la operación de reunión zeta bajo las condiciones siguientes.

   a. Sean L_1 y L_2 atributos de E_1 y de E_2, respectivamente. Supóngase que la condición de reunión θ implica únicamente los atributos de $L_1 \cup L_2$. Entonces,

$$\prod_{L_1 \cup L_2} (E_1 \bowtie_\theta E_2) =$$
$$= (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2} (E_2))$$

   b. Considérese una reunión $E_1 \bowtie_\theta E_2$. Sean L_1 y L_2 conjuntos de atributos de E_1 y de E_2, respectivamente. Sean L_3 los atributos de E_1 que están implicados en la condición de reunión θ, pero que no están en $L_1 \cup L_2$, y sean L_4 los atributos de E_2 que están implicados en la condición de reunión θ, pero que no están en $L_1 \cup L_2$. Entonces,

$$\Pi_{L_1 \cup L_2} (E_1 \bowtie_\theta E_2) =$$
$$= \Pi_{L_1 \cup L_2} ((\Pi_{L_1 \cup L_3} (E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4} (E_2)))$$

9. Las operaciones de conjuntos unión e intersección son conmutativas.

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

La diferencia de conjuntos no es conmutativa.

10. La unión y la intersección de conjuntos son asociativas.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. La operación de selección se distribuye por las operaciones de unión, intersección y diferencia de conjuntos.

$$\sigma_P (E_1 - E_2) = \sigma_P (E_1) - \sigma_P (E_2)$$

De manera parecida, la equivalencia anterior, con – sustituido por $\cup$ o por $\cap$, también es válida. Además,

$$\sigma_P (E_1 - E_2) = \sigma_P (E_1) - E_2$$

La equivalencia anterior, con – sustituido por ∩, también es válida, pero no se cumple si – se sustituye por ∪.

12. La operación de proyección se distribuye por la operación unión.

$$\Pi_L (E_1 \cup E_2) = (\Pi_L (E_1)) \cup (\Pi_L (E_2))$$

Ésta es sólo una lista parcial de las equivalencias. En los ejercicios se discuten más equivalencias que implican a los operadores relacionales extendidos, como la reunión externa y la agregación.

14.3.2. Ejemplos de transformaciones

Ahora se ilustrará el empleo de las reglas de equivalencia. Se utilizará el ejemplo del banco con los esquemas de relaciones:

Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activo*)
Esquema-cuenta = (*número-cuenta, nombre-sucursal, saldo*)
Esquema-impositor = (*nombre-cliente, número-cuenta*)

Las relaciones *sucursal*, *cuenta* e *impositor* son ejemplos de estos esquemas.

En el ejemplo del Apartado 14.1 la expresión

$$\Pi_{nombre\text{-}cliente} (\sigma_{ciudad\text{-}sucursal \,=\, \text{«Arganzuela»}} (sucursal \bowtie (cuenta \bowtie impositor)))$$

se transformó en la expresión siguiente,

$$\Pi_{nombre\text{-}cliente} ((\sigma_{ciudad\text{-}sucursal \,=\, \text{«Arganzuela»}} (sucursal)) \bowtie (cuenta \bowtie impositor))$$

que es equivalente a la expresión algebraica original, pero que genera relaciones intermedias de menor tamaño. Esta transformación se puede llevar a cabo empleando la regla 7.a. Recuérdese que la regla sólo dice que las dos expresiones son equivalentes; no dice que una sea mejor que la otra.

Se pueden utilizar varias reglas de equivalencia, una tras otra, sobre una consulta o sobre partes de una consulta. Como ejemplo, supóngase que se modifica la consulta original para restringir la atención a los clientes que tienen un saldo superior a 1.000 €. La nueva consulta del álgebra relacional es

$$\Pi_{nombre\text{-}cliente} (\sigma_{ciudad\text{-}sucursal \,=\, \text{«Arganzuela»} \,\wedge\, saldo\, >1000} (sucursal \bowtie (cuenta \bowtie impositor)))$$

No se puede aplicar el predicado de la selección directamente a la relación *sucursal*, ya que el predicado implica atributos tanto de la relación *sucursal* como de la relación *cuenta*. No obstante, se puede aplicar antes la regla 6.a (asociatividad de la reunión natural) para transformar la reunión *sucursal* ⋈ (*cuenta* ⋈ *impositor*) en (*sucursal* ⋈ *cuenta*) ⋈ *impositor*:

$$\Pi_{nombre\text{-}cliente} (\sigma_{ciudad\text{-}sucursal \,=\, \text{«Arganzuela»} \,\wedge\, saldo\, >1000} ((sucursal \bowtie cuenta) \bowtie impositor))$$

Luego, empleando la regla 7.a, se puede reescribir la consulta como

$$\Pi_{nombre\text{-}cliente} ((\sigma_{ciudad\text{-}sucursal \,=\, \text{«Arganzuela»} \,\wedge\, saldo>1000} (sucursal \bowtie cuenta)) \bowtie impositor)$$

Examinemos ahora la subexpresión de selección de esta expresión. Empleando la regla 1 se puede partir la selección en dos, para obtener la subexpresión siguiente:

$$\sigma_{ciudad\text{-}sucursal \,=\, \text{«Arganzuela»}} (\sigma_{saldo\, >1000} (sucursal \bowtie cuenta))$$

Las dos expresiones anteriores seleccionan tuplas con *ciudad-sucursal* = «Arganzuela» y *saldo* > 1000. Sin embargo, la última forma de la expresión ofrece una nueva oportunidad de aplicar la regla «llevar a cabo primero las selecciones», que da lugar a la subexpresión

$$\sigma_{ciudad\text{-}sucursal \,=\text{«Arganzuela»}} (sucursal) \bowtie \sigma_{saldo>1000} (cuenta)$$

La Figura 14.3 muestra la expresión inicial y la expresión final después de todas estas transformaciones. También se podría haber utilizado la regla 7.b para obtener directamente la expresión final, sin utilizar la regla 1 para partir la selección en dos selecciones. De hecho, la regla 7.b puede obtenerse de las reglas 1 y 7.a.

Se dice que un conjunto de reglas de equivalencia es **mínimo** si no se puede obtener ninguna regla a partir de una reunión de las demás. El ejemplo anterior muestra que el conjunto de reglas de equivalencia del Apartado 14.3.1 no es mínimo. Se puede generar una expresión equivalente a la original de diferentes maneras; el número de maneras diferentes de generar una expresión aumenta cuando se utiliza un conjunto de reglas de equivalencia que no es mínimo. Los optimizadores de consultas, por tanto, utilizan conjuntos mínimos de reglas de equivalencia.

Considérese ahora la siguiente forma de la consulta de ejemplo:

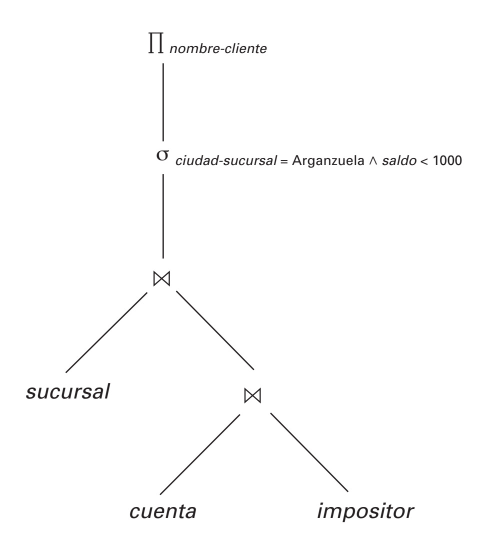
$$\Pi_{nombre\text{-}cliente} ((\sigma_{ciudad\text{-}sucursal \,=\, \text{«Arganzuela»}} (sucursal) \bowtie cuenta) \bowtie impositor)$$

Cuando se calcula la subexpresión

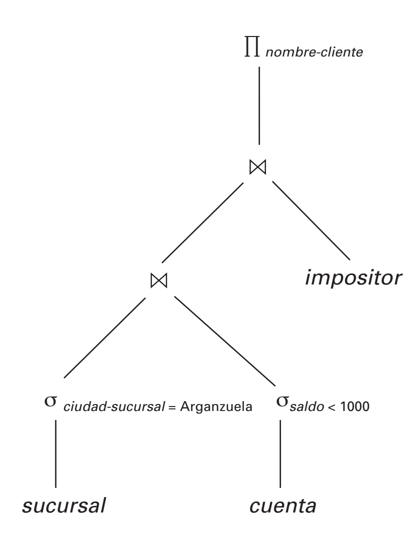
$$(\sigma_{ciudad\text{-}sucursal \,=\, \text{«Arganzuela»}} (sucursal) \bowtie cuenta)$$

se obtiene una relación cuyo esquema es

(*nombre-sucursal, ciudad-sucursal, activo, número-cuenta, saldo*)



(a) Árbol inicial de la expresión



(b) Árbol después de varias transformaciones

FIGURA 14.3. Varias transformaciones.

Se pueden eliminar varios atributos del esquema, forzando las proyecciones de acuerdo con las reglas de equivalencia 8.a y 8.b. Los únicos atributos que se deben conservar son los que aparecen en el resultado de la consulta y los que se necesitan para procesar las operaciones subsiguientes. Al eliminar los atributos innecesarios se reduce el número de columnas del resultado intermedio. Por tanto, se reduce el tamaño del resultado intermedio. En el ejemplo el único atributo que se necesita de la reunión de *sucursal* y de *cuenta* es *número-cuenta*.

Por tanto, se puede modificar la expresión hasta

$$\Pi_{nombre\text{-}cliente} ((\Pi_{número\text{-}cuenta} ((\sigma_{ciudad\text{-}sucursal = «Arganzuela»} (sucursal)) \bowtie cuenta)) \bowtie impositor)$$

La proyección $\Pi_{número\text{-}cuenta}$ reduce el tamaño de los resultados de las reuniones intermedias.

14.3.3. Ordenación de las reuniones

Una buena ordenación de las operaciones de reunión es importante para reducir el tamaño de los resultados temporales; por tanto, la mayor parte de los optimizadores de consultas prestan mucha atención al orden de las reuniones. Como se mencionó en el Capítulo 3 y en la regla de equivalencia 6.a, la operación de reunión natural es asociativa. Por tanto, para todas las relaciones r_1, r_2 y r_3,

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

Aunque estas expresiones sean equivalentes, los costes de calcular cada una de ellas pueden ser diferentes.

Considérese una vez más la expresión

$$\Pi_{nombre\text{-}cliente} ((\sigma_{ciudad\text{-}sucursal = «Arganzuela»} (sucursal)) \bowtie cuenta \bowtie impositor)$$

Se podría escoger calcular primero *cuenta* ⋈ *impositor* y luego combinar el resultado con

$$\sigma_{ciudad\text{-}sucursal = «Arganzuela»} (sucursal)$$

Sin embargo, es probable que *cuenta* ⋈ *impositor* sea una relación de gran tamaño, ya que contiene una tupla por cada cuenta. Por el contrario,

$$\sigma_{ciudad\text{-}sucursal = «Arganzuela»} (sucursal) \bowtie cuenta$$

es probablemente una relación de pequeño tamaño. Para comprobar que es así, se observa que, dado que el banco tiene un gran número de sucursales ampliamente distribuidas, es probable que sólo una pequeña parte de los clientes del banco tenga cuenta en las sucursales ubicadas en Arganzuela. Por tanto, la expresión anterior da lugar a una tupla por cada cuenta abierta por un residente de Arganzuela. Así, la relación temporal que se debe almacenar es menor que si se hubiera calculado primero *cuenta* ⋈ *impositor*.

Hay otras opciones a considerar a la hora de evaluar la consulta. No hay que preocuparse del orden en que aparecen los atributos en las reuniones, ya que resulta sencillo cambiarlo antes de mostrar el resultado. Por tanto, para todas las relaciones r_1 y r_2,

$$r_1 \bowtie r_2 = r_2 \bowtie r_1$$

Es decir, la reunión natural es conmutativa (regla de equivalencia 5).

Mediante la asociatividad y la conmutatividad de la reunión natural (reglas 5 y 6) se puede considerar reescribir la expresión del álgebra relacional como

$$\Pi_{nombre\text{-}cliente} (((\sigma_{ciudad\text{-}sucursal = «Arganzuela»} (sucursal)) \bowtie impositor) \bowtie cuenta)$$

Es decir, se puede calcular primero

$$(\sigma_{ciudad\text{-}sucursal\,=\,«Arganzuela»}\ (sucursal)) \bowtie\ impositor$$

y, luego, reunir el resultado con *cuenta*. Obsérvese, no obstante, que no hay atributos en común entre *Esquema-sucursal* y *Esquema-impositor*, por lo que la reunión no es más que un producto cartesiano. Si hay *o* sucursales en Arganzuela e *i* tuplas en la relación *impositor*, este producto cartesiano genera $o * i$ tuplas, una por cada par posible de tuplas de impositor y de sucursal (independientemente de si la cuenta de *impositor* está abierta en la sucursal). Por tanto, parece que este producto cartesiano producirá una relación temporal de gran tamaño. En consecuencia, esta estrategia se rechaza. No obstante, si el usuario ha introducido la expresión anterior, se pueden utilizar la asociatividad y la conmutatividad de la reunión natural para transformarla en la expresión más eficiente que se usó anteriormente.

14.3.4. Enumeración de expresiones equivalentes

Los optimizadores de consultas utilizan las reglas de equivalencia para generar de manera sistemática expresiones equivalentes a la expresión de consulta dada. Conceptualmente el proceso se desarrolla de la manera siguiente. Dada una expresión, si alguna subexpresión coincide con algún lado de una regla de equivalencia, el optimizador genera una nueva expresión en que la subexpresión se transforma para coincidir con el otro lado de la regla. Este proceso continúa hasta que no se puedan generar expresiones nuevas.

El proceso anterior resulta costoso tanto en espacio como en tiempo. Éste es el modo en que se puede reducir los requisitos de espacio: si se genera una expresión E_1 a partir de una expresión E_2 empleando una regla de equivalencia, E_1 y E_2 son parecidas en estructura y tienen subexpresiones que son idénticas. Las técnicas de representación de expresiones que permiten que las dos expresiones apunten a las subexpresiones compartidas pueden reducir de manera significativa los requisitos de espacio, y muchos optimizadores de consultas las utilizan.

Además, no siempre resulta necesario generar todas las expresiones que pueden generarse con las reglas de equivalencia. Si un optimizador tiene en cuenta las estimaciones de costes, puede que logre evitar el examen de algunas de las expresiones, como se verá en el Apartado 14.4. Se puede reducir el tiempo necesario para la optimización empleando técnicas como éstas.

14.4. ELECCIÓN DE LOS PLANES DE EVALUACIÓN

La generación de expresiones sólo es una parte del proceso de optimización de consultas, ya que cada operación de la expresión puede implementarse con algoritmos diferentes. Por tanto, se necesita un plan de evaluación para definir exactamente el algoritmo que se utilizará para cada operación y el modo en que se coordinará la ejecución de las operaciones. La Figura 14.4 muestra un plan de evaluación posible para la expresión de la Figura 14.3. Como ya se ha visto, se pueden emplear varios algoritmos diferentes para cada operación relacional, lo que da lugar a planes de evaluación alternativos. Además, hay que tomar decisiones sobre el encauzamiento. En la figura, los trazos de las operaciones de selección hasta la operación de reunión mezcla están marcados como encauzados; el encauzamiento es factible si las operaciones de selección generan sus resultados ordenados según los atributos de reunión. Lo harán si los índices de *sucursal* y *cuenta* almacenan los registros con valores iguales de los atributos de índice ordenados por *nombre-sucursal*.

14.4.1. Interacción de las técnicas de evaluación

Una manera de escoger un plan de evaluación para una expresión de consulta es sencillamente escoger para cada operación el algoritmo más económico para evaluarla. Se puede escoger cualquier ordenación de las operaciones que asegure que las operaciones ubicadas por debajo en el árbol se ejecuten antes que las operaciones situadas más arriba.

Sin embargo, la selección del algoritmo más económico para cada operación no es necesariamente una buena idea. Aunque puede que una reunión por mezcla en un nivel dado resulte más costosa que una reunión por

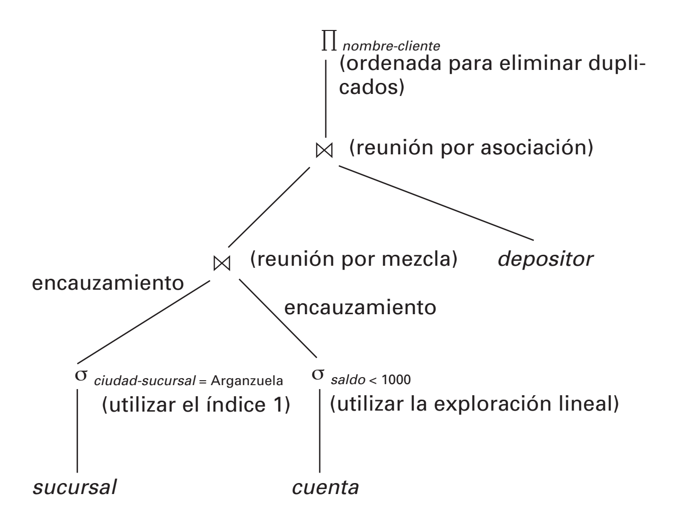


FIGURA 14.4. Un plan de evaluación.

asociación, puede que proporcione un resultado ordenado que haga más económica la evaluación de operaciones posteriores (como la eliminación de duplicados, la intersección u otra reunión por asociación). De manera parecida, puede que una reunión en bucle anidado indexada proporcione oportunidades para el encauzamiento de los resultados de la operación siguiente y, por tanto, puede que resulte útil aunque no sea la manera más económica de llevar a cabo la reunión. Para escoger el mejor algoritmo global hay que considerar incluso los algoritmos no óptimos para operaciones individuales.

Por tanto, además de considerar las expresiones alternativas de cada consulta, también hay que considerar los algoritmos alternativos para cada operación de cada expresión. Se pueden utilizar reglas muy parecidas a las reglas de equivalencia para definir los algoritmos que pueden utilizarse para cada operación, y si su resultado puede encauzarse o se debe materializar. Se pueden utilizar estas reglas para generar todos los planes de evaluación de consultas para una expresión dada.

Dado un plan de evaluación, se puede estimar su coste empleando las estadísticas estimadas mediante las técnicas del Apartado 14.2 junto con las estimaciones de costes de varios algoritmos y métodos de evaluación descritos en el Capítulo 13. Eso sigue dejando el problema de la selección del mejor plan de evaluación de la consulta. Hay dos enfoques generales: el primero busca todos los planes y escoge el mejor de una manera basada en los costes. El segundo utiliza la heurística para escoger el plan. A continuación se estudiarán los dos enfoques. Los optimizadores de consultas prácticos incorporan elementos de ambos enfoques.

14.4.2. Optimización basada en el coste

Los **optimizadores basados en el coste** generan una gama de planes de evaluación a partir de la consulta dada empleando las reglas de equivalencia y escogen el de coste mínimo. Para las consultas complejas el número de planes de consulta diferentes que son equivalentes a un plan dado puede ser grande. A modo de ejemplo, considérese la expresión

$$r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$$

en la que las reuniones se expresan sin ninguna ordenación. Con $n = 3$, hay 12 diferentes ordenaciones de la mezcla:

$$\begin{array}{ll} r_1 \bowtie (r_2 \bowtie r_3) & r_1 \bowtie (r_3 \bowtie r_2) \\ r_2 \bowtie (r_1 \bowtie r_3) & r_2 \bowtie (r_3 \bowtie r_1) \\ r_3 \bowtie (r_1 \bowtie r_2) & r_3 \bowtie (r_2 \bowtie r_1) \\ & \\ (r_2 \bowtie r_3) \bowtie r_1 & (r_3 \bowtie r_2) \bowtie r_1 \\ (r_1 \bowtie r_3) \bowtie r_2 & (r_3 \bowtie r_1) \bowtie r_2 \\ (r_1 \bowtie r_2) \bowtie r_3 & (r_2 \bowtie r_1) \bowtie r_3 \end{array}$$

En general, con n relaciones, hay $(2(n-1))!/(n-1)!$ órdenes de reunión diferentes. (Se deja el cálculo de esta expresión al lector en el Ejercicio 14.10.) Para las reuniones que implican números pequeños de relaciones, este número resulta aceptable; por ejemplo, con $n = 5$, el número es 1680. Sin embargo, a medida que n se incrementa, este número crece rápidamente. Con $n = 7$, el número es 665.280; con $n = 10$, ¡el número es mayor de 17.600 millones!

Afortunadamente, no es necesario generar todas las expresiones equivalentes a la expresión dada. Por ejemplo, supóngase que se desea hallar el mejor orden de reunión de la forma

$$(r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$$

que representa todos los órdenes de reunión en que r_1, r_2 y r_3 se reúnen primero (en algún orden), y el resultado se reúne (en algún orden) con r_4 y r_5. Hay doce órdenes de reunión diferentes para calcular $r_1 \bowtie r_2 \bowtie r_3$, y otros doce órdenes para calcular la reunión de este resultado con r_4 y r_5. Por tanto, parece que hay 144 órdenes de reunión que examinar. Sin embargo, una vez hallado el mejor orden de reunión para el subconjunto de relaciones $\{r_1, r_2, r_3\}$, se puede utilizar ese orden para las reuniones posteriores con r_4 y r_5, y se pueden ignorar todos los órdenes de reunión más costosos de $r_1 \bowtie r_2 \bowtie r_3$. Por tanto, en lugar de 144 opciones que examinar sólo hace falta examinar 12 + 12 opciones.

```
procedure hallarmejorplan(S)
    if (mejorplan[S].coste ≠ ∞)
        return mejorplan[S]
        if (S contiene sólo una relación)
            establecer mejorplan[S].plan y mejorplan[S].coste en términos de la mejor forma de acceder a S
    // mejorplan[S] no se ha calculado anteriormente, hay que calcularlo ahora
    else for each subconjunto no vacío S1 de S tal que S1 ≠ S
        P1 = hallarmejorplan(S1)
        P2 = hallarmejorplan(S – S1)
        A = mejor algoritmo para reunir los resultados de P1 y P2
        coste = P1.coste + P2.coste + coste de A
        if coste < mejorplan[S].coste
            mejorplan[S].coste = coste
            mejorplan[S].plan = «ejecutar P1.plan; ejecutar P2.plan; reunir resultados de P1 y de P2 utilizando A»
  return mejorplan[S]
```

FIGURA 14.5. Algoritmo de programación dinámica para la optimización del orden de reunión.

Utilizando esta idea se puede desarrollar un *algoritmo de programación dinámica* para hallar los órdenes de reunión óptimos. Los algoritmos de programación dinámica almacenan los resultados de los cálculos y los reutilizan, un procedimiento que puede reducir enormemente el tiempo de ejecución. En la Figura 14.5 aparece un procedimiento recursivo que implementa el algoritmo de programación dinámica.

El procedimiento almacena los planes de evaluación que calcula en el array asociado *mejorplan*, que está indexado por conjuntos de relaciones. Cada elemento del array asociativo contiene dos componentes: el coste del mejor plan de *S* y el propio plan. El valor de *mejorplan*[*S*].*coste* se supone que se inicializa como ∞ si *mejorplan*[*S*] no se ha calculado todavía.

El procedimiento comprueba en primer lugar si el mejor plan para calcular la reunión del conjunto de relaciones dado *S* se ha calculado ya (y se ha almacenado en el array asociativa *mejorplan*); si es así, devuelve el plan ya calculado. Si *S* sólo contiene una relación, se calcula la mejor forma de acceder a *S* (teniendo en cuenta las relaciones sobre *S*, si las hay) y se almacena en *mejorplan*. En caso contrario, el procedimiento intenta todas las maneras posibles de dividir *S* en dos subconjuntos disjuntos. Para cada división el procedimiento halla de manera recursiva los mejores planes para cada uno de los dos subconjuntos y luego calcula el coste del plan global utilizando esa división. El procedimiento escoge el plan más económico de entre todas las alternativas para dividir *S* en dos conjuntos. El procedimiento almacena el plan más económico y su coste en el array *mejorplan* y los devuelve. La complejidad temporal del procedimiento puede demostrarse que es $O(3^n)$ (véase el Ejercicio 14.11).

En realidad, el orden en que la reunión de un conjunto de relaciones genera las tuplas también es importante para hallar el mejor orden global de reunión, ya que puede afectar al coste de las reuniones posteriores (por ejemplo, si se utiliza una mezcla). Se dice que un orden determinado de las tuplas es un **orden interesante** si puede resultar útil para alguna operación posterior. Por ejemplo, la generación del resultado de $r_1 \bowtie r_2 \bowtie r_3$ ordenado según los atributos comunes con r_4 o r_5 puede resultar útil, pero generarlo ordenado según los atributos comunes solamente con r_1 y r_2 no resulta útil. El empleo de la reunión por mezcla para calcular $r_1 \bowtie r_2 \bowtie r_3$ puede resultar más costoso que emplear algún otro tipo de técnica de reunión, pero puede que proporcione un resultado ordenado según un orden interesante.

Por tanto, no basta con hallar el mejor orden de reunión para cada subconjunto del conjunto de *n* relaciones dadas. Por el contrario, hay que hallar el mejor orden de reunión para cada subconjunto para cada orden interesante de la reunión resultante para ese subconjunto. El número de subconjuntos de *n* relaciones es 2^n. El número de órdenes de colocación interesantes no suele ser grande. Así, hay que almacenar alrededor de 2^n expresiones de reunión. El algoritmo de programación dinámica para hallar el mejor orden de reunión puede extenderse de manera sencilla para que trabaje con los órdenes de colocación. El coste del algoritmo extendido depende del número de órdenes interesantes para cada subconjunto de relaciones; dado que se ha hallado que este número en la práctica es pequeño, el coste se queda en $O(3^n)$.

Con $n = 10$ este número es de alrededor de 59.000, que es mucho mejor que los 17.600 millones de órdenes de reunión diferentes. Y lo que es más importante, el almacenamiento necesario es mucho menor que antes, ya que sólo hace falta almacenar un orden de reunión por cada orden interesante de cada uno de los 1.024 subconjuntos de $r_1, \dots, r_{10}$. Aunque los dos números siguen creciendo rápidamente con *n*, las reuniones que se producen con frecuencia suelen tener menos de diez relaciones y pueden manejarse con facilidad.

Se pueden utilizar varias técnicas para reducir aún más el coste de la búsqueda entre un gran número de planes. Por ejemplo, al examinar los planes para una expresión se puede concluir tras examinar sólo una parte de la expresión si se determina que el plan más económico para esa parte ya resulta más costoso que el plan de evaluación más económico para una expresión completa examinada anteriormente. De manera parecida, supóngase que se determina que la manera más económica de evaluar una subexpresión es más costosa que el plan de evaluación más económico para una expresión completa examinada anteriormente. Entonces, no hace falta examinar ninguna expresión completa que incluya esa subexpresión. Se puede reducir aún más el número de planes de evaluación que hay que considerar completamente llevando a cabo antes una selección heurística de un buen plan y estimando el coste de ese plan. Luego, sólo unos pocos planes competidores necesitarán un análisis completo de los costes. Estas optimizaciones pueden reducir de manera significativa la sobrecarga de la optimización de consultas.

14.4.3. Optimización heurística

Un inconveniente de la optimización basada en el coste es el coste de la propia optimización. Aunque el coste del procesamiento de las consultas puede reducirse mediante optimizaciones inteligentes, la optimización basada en el coste sigue resultando costosa. Por ello, muchos sistemas utilizan la **heurística** para reducir el número de elecciones que hay que hacer de una manera basada en los costes. Algunos sistemas incluso deciden utilizar sólo la heurística y no utilizan en absoluto la optimización basada en el coste.

Un ejemplo de regla heurística es la siguiente regla para la transformación de consultas del álgebra relacional:

- Llevar a cabo las operaciones de selección tan pronto como sea posible.

Los optimizadores heurísticos utilizan esta regla sin averiguar si se reduce el coste mediante esta transformación.

En el primer ejemplo de transformación del Apartado 14.3 se forzó la operación de selección en una reunión.

Se dice que la regla anterior es heurística porque suele ayudar a reducir el coste, aunque no lo haga siempre. Como ejemplo de dónde puede dar lugar a un incremento del coste, considérese una expresión $\sigma_\theta (r \bowtie s)$, donde la condición θ sólo hace referencia a atributos de s. Ciertamente, la selección puede llevarse a cabo antes de la reunión. Sin embargo, si r es tremendamente pequeña comparada con s, y si hay un índice basado en los atributos de reunión de s pero no hay ningún índice basado en los atributos utilizados por θ, probablemente resulte una mala idea llevar a cabo la selección pronto. Llevar a cabo pronto la selección —es decir, directamente sobre s— exigiría hacer una exploración de todas las tuplas de s. Probablemente resulte más económico calcular la reunión utilizando el índice y luego rechazar las tuplas que no superen la selección.

La operación de proyección, como la operación de selección, reduce el tamaño de las relaciones. Por tanto, siempre que haya que generar una relación temporal, resulta ventajoso aplicar inmediatamente cuantas proyecciones sea posible. Esta ventaja sugiere un acompañante a la heurística «llevar a cabo las selecciones tan pronto como sea posible»:

- Llevar a cabo las proyecciones tan pronto como sea posible.

Suele resultar mejor llevar a cabo las selecciones antes que las proyecciones, ya que las selecciones tienen la posibilidad de reducir mucho el tamaño de las relaciones y permiten el empleo de índices para tener acceso a las tuplas. Un ejemplo parecido al utilizado para la heurística de selección debería convencer al lector de que esta heurística no siempre reduce el coste.

Aprovechando las equivalencias estudiadas en el Apartado 14.3.1, un algoritmo de optimización heurística reordenará los componentes de un árbol de consultas inicial para conseguir una ejecución mejorada de la consulta. A continuación se presenta una visión general de las etapas de un algoritmo típico de optimización heurística. El lector comprenderá la heurística al visualizar la expresión de la consulta como si fuera un árbol, como se muestra en la Figura 14.3.

1. Hay que descomponer las selecciones conjuntivas en una secuencia de operaciones de selección sencillas. Este paso, basado en la regla de equivalencia 1, facilita el desplazamiento de las operaciones de selección hacia la parte inferior del árbol de consultas.
2. Hay que desplazar las operaciones de selección hacia la parte inferior del árbol de consultas para conseguir su ejecución lo antes posible. Este paso utiliza las propiedades de conmutatividad y de distributividad de la operación de selección puestas de manifiesto en las reglas de equivalencia 2, 7.a, 7.b y 11.

   Por ejemplo, este paso transforma $\sigma_\theta (r \bowtie s)$ en $\sigma_\theta (r) \bowtie s$ o en $r \bowtie \sigma_\theta (s)$ siempre que sea posible. La realización de las selecciones basadas en valores tan pronto como sea posible reduce el coste de ordenación y de mezcla de los resultados intermedios. El grado de reordenación permitido para una selección concreta viene determinado por los atributos implicados en esa condición de selección.
3. Hay que determinar las operaciones de selección y de reunión que producirán las relaciones de menor tamaño, es decir, que producirán las relaciones con el menor número de tuplas. Utilizando la asociatividad de la operación hay que reordenar el árbol para que las relaciones de los nodos hojas con esas selecciones restrictivas se ejecuten antes.

   Este paso considera la selectividad de las condiciones de selección o de reunión. Hay que recordar que la selección más restrictiva —es decir, la condición con la selectividad de menor tamaño— recupera el menor número de registros. Este paso confía en la asociatividad de las operaciones binarias dada en la regla de equivalencia 6.
4. Hay que sustituir por operaciones de reunión las operaciones producto cartesiano seguidas de condiciones de selección (regla 4.a). La operación producto cartesiano suele resultar costosa de implementar, ya que $r_1 \times r_2$ incluye un registro por cada combinación de registros procedentes de r_1 y de r_2. La selección puede reducir de manera significativa el número de registros, haciendo la reunión mucho menos costosa que el producto cartesiano.
5. Hay que dividir las listas de atributos de proyección y desplazarlas hacia la parte inferior del árbol todo lo que sea posible, creando proyecciones nuevas donde sea necesario. Este paso se aprovecha de las propiedades de la operación de proyección dadas en las reglas de equivalencia 3, 8.a, 8.b y 12.
6. Hay que identificar los subárboles cuyas operaciones pueden encauzarse y ejecutarlos utilizando el encauzamiento.

En resumen, las heurísticas citadas aquí reordenan la representación inicial del árbol de consultas de manera que las operaciones que reducen el tamaño de los resultados intermedios se apliquen antes; las selecciones tempranas reducen el número de tuplas y las proyecciones tempranas reducen el número de atributos. Las transformaciones heurísticas también reestructuran el árbol de modo que el sistema lleve a cabo las operaciones de selección y de reunión más restrictivas antes que otras operaciones parecidas.

La optimización heurística hace corresponder aún más la expresión de consulta transformada heurística-

mente con secuencias alternativas de operaciones para producir un conjunto candidato de planes de evaluación. Cada plan de evaluación no sólo incluye las operaciones relacionales que hay que llevar a cabo, sino también los índices que hay que emplear, el orden en el que hay que tener acceso a las tuplas y el orden en que hay que realizar las operaciones. La fase de **selección del plan de acceso** del optimizador heurístico selecciona la estrategia más eficiente para cada operación.

14.4.4. Estructura de los optimizadores de consultas **

Hasta ahora se han descrito dos enfoques básicos de la selección de planes de evaluación; como se ha indicado, los optimizadores de consultas más prácticos combinan elementos de ambos enfoques. Por ejemplo, algunos optimizadores de consultas, como el optimizador System R, no toman en consideración todos los órdenes de reunión, sino que restringen la búsqueda a tipos concretos de órdenes de reunión. El optimizador System R sólo toma en consideración los órdenes de reunión en que el operando de la derecha de cada reunión es una de las relaciones iniciales $r_1, \dots, r_n$. Estos órdenes de reunión se denominan **órdenes de reunión en profundidad por la izquierda**. Los órdenes de reunión en profundidad por la izquierda resultan especialmente convenientes para la evaluación encauzada, ya que el operando de la derecha es una relación almacenada y, así, sólo se encauza una entrada por cada reunión.

La Figura 14.6 muestra la diferencia entre un árbol de reunión en profundidad por la izquierda y otro que no lo es. El tiempo que se tarda en tomar en consideración todos los órdenes de reunión en profundidad por la izquierda es $O(n!)$, que es mucho menor que el tiempo necesario para tomar en consideración todos los órdenes de reunión. Con el empleo de las optimizaciones de programación dinámica, el optimizador de System R puede hallar el mejor orden de reunión en un tiempo de $O(n2^n)$. Compárese este coste con el tiempo de $O(3^n)$ necesario para hallar el mejor orden de reunión global. El optimizador System R utiliza la heurística para forzar el desplazamiento de las selecciones y de las proyecciones hacia la parte inferior del árbol de consultas.

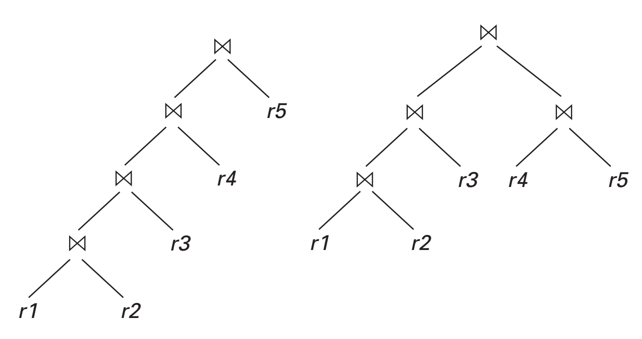


FIGURA 14.6. Árboles de reunión.

La estimación de costes que se presentó para la exploración mediante índices secundarios daba por supuesto que cada acceso a una tupla da lugar a una operación de E/S. Es probable que la estimación resulte precisa con memorias intermedias de pequeño tamaño; con memorias intermedias de gran tamaño, sin embargo, puede que la página que contiene la tupla esté ya en la memoria intermedia. Algunos optimizadores incorporan una técnica mejor de estimación de costes para estas exploraciones: tienen en cuenta la probabilidad de que la página que contiene la tupla se halle en el memoria intermedia.

Los enfoques de la optimización de consultas que integran la selección heurística y la generación de planes de acceso alternativos se han adoptado en varios sistemas. El enfoque utilizado en System R y en su sucesor, el proyecto Starburst, es un procedimiento jerárquico basado en el concepto de bloques anidados de SQL. Las técnicas de optimización basadas en costes aquí descritas se utilizan por separado para cada bloque de la consulta.

El enfoque heurístico de algunas versiones de Oracle funciona aproximadamente de esta manera: para cada reunión de grado *n* toma en consideración *n* planes de evaluación. Cada plan utiliza un orden de reunión en profundidad por la izquierda, comenzando con una relación diferente de las *n* existentes. La heurística crea el orden de reunión para cada uno de los *n* planes de evaluación seleccionando de manera repetida la «mejor» relación que reunir a continuación, con base en la clasificación de los caminos de acceso disponibles. Se escoge la reunión en bucle anidado o mezcla-ordenación para cada una de las reuniones, en función de los caminos de acceso disponibles. Finalmente, la heurística escoge uno de los *n* planes de evaluación de manera heurística, basada en la minimización del número de reuniones de bucle anidado que no tienen disponible un índice para la relación interna y en el número de reuniones por mezcla-ordenación.

La complejidad de SQL introduce un elevado grado de complejidad en los optimizadores de consultas. En concreto, resulta difícil traducir las subconsultas anidadas de SQL al álgebra relacional. Se describirá brevemente el modo de tratar las subconsultas anidadas en el Apartado 14.4.5. Para las consultas compuestas de SQL (que utilizan la operación ∪, ∩ o –), el optimizador procesa cada componente por separado y combina los planes de evaluación para formar el plan global de evaluación.

Incluso con el uso de la heurística la optimización de consultas basada en los costes impone una sobrecarga sustancial al procesamiento de las consultas. No obstante, el coste añadido de la optimización de las consultas basada en los costes suele compensarse con creces por el ahorro en tiempo de ejecución de la consulta, que queda dominado por los accesos lentos a los

discos. La diferencia en el tiempo de ejecución entre un buen plan y uno malo puede ser enorme, lo que vuelve esencial la optimización de las consultas. El ahorro conseguido se multiplica en las aplicaciones que se ejecutan de manera regular, en las que se puede optimizar la consulta una sola vez y utilizarse el plan de consultas seleccionado en cada ejecución. Por tanto, la mayor parte de los sistemas comerciales incluyen optimizadores relativamente sofisticados. Las notas bibliográficas dan referencias de las descripciones de los optimizadores de consultas de los sistemas de bases de datos reales.

14.4.5. Optimización de las subconsultas anidadas**

SQL trata conceptualmente a las subconsultas anidadas de la cláusula **where** como funciones que toman parámetros y devuelven un solo valor o un conjunto de valores (quizás, un conjunto vacío). Los parámetros son las variables de la consulta del nivel externo que se utilizan en la subconsulta anidada (estas variables se denominan **variables de correlación**). Por ejemplo, supóngase que se tiene la consulta siguiente.

select *nombre-cliente*
from *prestatario*
where exists (**select** *
  from *impositor*
  where *impositor.nombre-cliente* =
    = *prestatario.nombre-cliente*)

Conceptualmente, la subconsulta puede considerarse como una función que toma un parámetro (aquí, *prestatario.nombre-cliente*) y devuelve el conjunto de todos los impositores con el mismo nombre.

SQL evalúa la consulta global (conceptualmente) calculando el producto cartesiano de las relaciones de la cláusula **from** externa y comprobando luego los predicados de la cláusula **where** para cada tupla del producto. En el ejemplo anterior, el predicado comprueba si el resultado de la evaluación de la subconsulta está vacío.

Esta técnica para evaluar una consulta con una subconsulta anidada se denomina **evaluación correlacionada**. La evaluación correlacionada no resulta muy eficiente, ya que la subconsulta se evalúa por separado para cada tupla de la consulta del nivel externo. Puede dar lugar a gran número de operaciones aleatorias de E/S a disco.

Por tanto, los optimizadores de SQL intentan transformar las subconsultas anidadas en reuniones, siempre que resulta posible. Los algoritmos de reunión eficientes evitan las costosas operaciones aleatorias de E/S. Cuando la transformación no resulta posible el optimizador conserva las subconsultas como expresiones independientes, las optimiza por separado y luego las evalúa mediante la evaluación correlacionada.

Como ejemplo de transformación de una subconsulta anidada en una reunión, la consulta del ejemplo anterior puede reescribirse como:

select *nombre-cliente*
from *prestatario*, *impositor*
where *impositor.nombre-cliente* =
  = *prestatario.nombre-cliente*

(Para reflejar correctamente la semántica de SQL no debe cambiar el número de derivaciones duplicadas debido a la reescritura; la consulta reescrita puede modificarse para asegurarse de que se cumple esta propiedad, como se verá en breve.)

En el ejemplo la subconsulta anidada era muy sencilla. En general, puede que no resulte posible desplazarlas relaciones de la subconsulta anidada a la cláusula **from** de la consulta externa. En lugar de eso, se crea una relación temporal que contiene el resultado de la consulta anidada *sin* las selecciones empleando las variables de correlación de la consulta externa y se reúne la tabla temporal con la consulta del nivel exterior. Por ejemplo, una consulta de la forma

select …
from L_1
where P_1 **and exists** (**select** *
  from L_2
  where P_2)

donde P_2 es una conjunción de predicados más sencillos, puede reescribirse como

create table t_1 **as**
  select distinct V
    from L_2
    where P_2^1
select …
from L_1, t_1
where P_1 **and** P_2^2

donde P_2^1 contiene los predicados de P_2 sin las selecciones que implican las variables de correlación, y P_2^2 reintroduce las selecciones que implican las variables de correlación (con las relaciones a que se hace referencia en el predicado renombradas adecuadamente). Aquí V contiene todos los atributos que se utilizan en las selecciones con las variables de correlación en la subconsulta anidada.

En el ejemplo la consulta original se habría transformado en

create table t_1 **as**
  select distinct *nombre-cliente*
  from *impositor*
select *nombre-cliente*
from *prestatario*, t_1
where t_1.*nombre-cliente* = *prestatario.nombre-cliente*

La consulta que se reescribió para mostrar la creación de relaciones temporales puede obtenerse simplificando la consulta transformada más arriba, suponiendo que el número de duplicados de cada tupla no importa.

El proceso de sustituir una consulta anidada por una consulta con una reunión (acaso con una relación temporal) se denomina **descorrelación**.

La descorrelación resulta más complicada cuando la subconsulta anidada utiliza la agregación o cuando el resultado de la subconsulta anidada se utiliza para comprobar la igualdad, o cuando la condición que enlaza la subconsulta anidada con la consulta exterior es **not exists**, etcétera. No se intentará dar algoritmos para el caso general y, en vez de eso, se remitirá al lector a los elementos de importancia en las notas bibliográficas.

La optimización de las subconsultas anidadas complejas es una labor difícil, como puede deducirse del estudio anterior, y muchos optimizadores sólo llevan a cabo una cantidad limitada de descorrelación. Resulta más conveniente evitar el empleo de subconsultas anidadas complejas, siempre que sea posible, ya que no se puede asegurar que el optimizador de consultas tenga éxito en su conversión a una forma que pueda evaluarse de manera eficiente.

14.5. VISTAS MATERIALIZADAS**

Cuando se define una vista, normalmente la base de datos sólo almacena la consulta que define la vista. Por el contrario, una **vista materializada** es una vista cuyo contenido se calcula y se almacena. Las vistas materializadas constituyen datos redundantes, en el sentido de que su contenido puede deducirse de la definición de la vista y del resto del contenido de la base de datos. No obstante, resulta mucho más económico en muchos casos leer el contenido de una vista materializada que calcular el contenido de la vista ejecutando la consulta que la define.

Las vistas materializadas resultan importantes para la mejora del rendimiento de algunas aplicaciones. Considérese esta vista, que da el importe total de los préstamos de cada sucursal:

create view *total-préstamos-sucursal*
　　　　(*nombre-sucursal*, *total-préstamos*) **as**
select *nombre-sucursal*, **sum**(*importe*)
from *préstamos*
groupby *nombre-sucursal*

Supóngase que el importe total de los préstamos de la sucursal se solicita con frecuencia (antes de conceder un nuevo préstamo, por ejemplo). El cálculo de la vista exige la lectura de cada tupla de *préstamos* correspondiente a la sucursal y sumar los importes de los préstamos, lo que puede ocupar mucho tiempo. Por el contrario, si la definición de la vista del importe total de los préstamos estuviera materializada, el importe total de los préstamos podría hallarse buscando una sola tupla de la vista materializada.

14.5.1. Mantenimiento de las vistas

Un problema con las vistas materializadas es que hay que mantenerlas actualizadas cuando se modifican los datos empleados en la definición de la vista. Por ejemplo, si se actualiza el valor *importe* de un préstamo, la vista materializada se vuelve inconsistente con los datos subyacentes y hay que actualizarla. La tarea de mantener actualizada una vista materializada con los datos subyacentes se denomina **mantenimiento de la vista**.

Las vistas pueden mantenerse mediante código escrito a mano: es decir, cada fragmento del código que actualiza el valor *importe* del préstamo puede modificarse para actualizar el importe total de los préstamos de la sucursal correspondiente.

Otra opción para el mantenimiento de las vistas materializadas es la definición de desencadenadores para la inserción, la eliminación y la actualización de cada relación de la definición de la vista. Los desencadenadores deben modificar el contenido de la vista materializada para tener en cuenta el cambio que ha provocado que se active el desencadenador. Una manera simplista de hacerlo es volver a calcular completamente la vista materializada con cada actualización.

Una opción mejor es modificar sólo las partes afectadas de la vista materializada, lo que se conoce como **mantenimiento incremental de la vista**. En el Apartado 14.5.2 se describe la manera de llevar a cabo el mantenimiento incremental de la vista.

Los sistemas modernos de bases de datos proporcionan más soporte directo para el mantenimiento incremental de las vistas. Los programadores de bases de datos ya no necesitan definir desencadenadores para el mantenimiento de las vistas. Por el contrario, una vez que se ha declarado materializada una vista, el sistema de bases de datos calcula su contenido y actualiza de manera incremental el contenido cuando se modifican los datos subyacentes.

14.5.2. Mantenimiento incremental de las vistas

Para comprender el modo de mantener de manera incremental las vistas materializadas se comenzará por considerar las operaciones individuales y luego se verá la manera de manejar una expresión completa. Los cambios de cada relación que puedan hacer que se quede

desactualizada una vista materializada son las inserciones, las eliminaciones y las actualizaciones. Para simplificar la descripción se sustituyen las actualizaciones a cada tupla por la eliminación de esa tupla seguida de la inserción de la tupla actualizada. Por tanto, sólo hay que considerar las inserciones y las eliminaciones. Los cambios (inserciones y eliminaciones) en la relación o en la expresión se denominan su **diferencial**.

14.5.2.1. La operación reunión

Considérese la vista materializada $v = r \bowtie s$. Supóngase que se modifica r insertando un conjunto de tuplas denotado por i_r. Si el valor antiguo de r se denota por r^{vieja} y el valor nuevo de r por r^{nueva}, $r^{nueva} = r^{vieja} \cup i_r$. Ahora bien, el valor antiguo de la vista, v^{vieja} viene dado por $r^{vieja} \bowtie s$, y el valor nuevo v^{nueva} viene dado por $r^{nueva} \bowtie s$. Se puede reescribir $r^{nueva} \bowtie s$ como $(r^{vieja} \cup i_r) \bowtie s$, lo que se puede reescribir una vez más como $(r^{vieja} \bowtie s) \cup (i_r \bowtie 1\ s)$. En otros términos,

$$v^{nueva} = v^{vieja} \cup (i_r \bowtie s)$$

Por tanto, para actualizar la vista materializada v, sólo hace falta añadir las tuplas $i_r \bowtie s$ al contenido antiguo de la vista materializada. Las inserciones en s se manejan de una manera completamente simétrica.

Supóngase ahora que se modifica r eliminando un conjunto de tuplas denotado por d_r. Utilizando el mismo razonamiento que anteriormente se obtiene

$$v^{nueva} = v^{vieja} - (d_r \bowtie s)$$

Las eliminaciones en s se manejan de una manera completamente simétrica.

14.5.2.2. Las operaciones selección y proyección

Considérese una vista $v = \sigma_\theta(r)$. Si se modifica r insertando un conjunto de tuplas i_r, el valor nuevo de v puede calcularse como

$$v^{nueva} = v^{vieja} \cup \sigma_\theta(i_r)$$

De manera parecida, si se modifica r eliminando un conjunto de tuplas e_r, el valor nuevo de v puede calcularse como

$$v^{nueva} = v^{vieja} - \sigma_\theta(e_r)$$

La proyección es una operación más difícil de tratar. Hay que considerar una vista materializada $v = \Pi_A(r)$. Supóngase que la relación r está en el esquema $R = (A, B)$ y que r contiene dos tuplas, $(a, 2)$ y $(a, 3)$. Entonces, $\Pi_A(r)$ tiene una sola tupla, (a). Si se elimina la tupla $(a, 2)$ de r, no se puede eliminar la tupla (a) de $\Pi_A(r)$: si se hiciera, el resultado sería una relación vacía, mientras que en realidad $\Pi_A(r)$ sigue teniendo una tupla (a). El motivo es que la misma tupla (a) se obtiene de dos maneras, y que la eliminación de una tupla de r sólo elimina una de las formas de obtener (a); la otra sigue presente.

Este motivo también ofrece una pista de la solución: para cada tupla de una proyección como $\Pi_A(r)$, se lleva la cuenta del número de veces que se ha obtenido.

Cuando se elimina un conjunto de tuplas e_r de r, para cada tupla t de e_r hay que hacer lo siguiente.

$t.A$ denota la proyección de t sobre el atributo A. Se busca $(t.A)$ en la vista materializada y se disminuye la cuenta almacenada con ella en 1. Si la cuenta llega a 0, se elimina $(t.A)$ de la vista materializada.

El manejo de las inserciones resulta relativamente directo. Cuando un conjunto de tuplas i_r se inserta en r, para cada tupla t de i_r se hace lo siguiente. Si $(t.A)$ ya está presente en la vista materializada, se incrementa la cuenta almacenada con ella en 1. En caso contrario, se añade $(t.A)$ a la vista materializada con la cuenta definida como 1.

14.5.2.3. Las operaciones de agregación

Las operaciones de agregación se comportan aproximadamente como las proyecciones. Las operaciones de agregación en SQL son **count, sum, avg, min** y **max**:

- **count**: Considérese una vista materializada $v = {}_A\mathcal{G}_{cuenta(B)}(r)$, que calcula la cuenta del atributo B, después de agrupar r según el atributo A.

  Cuando se inserta un conjunto de tuplas i_r en r, para cada tupla t de i_r hay que hacer lo siguiente. Se busca el grupo $t.A$ en la vista materializada. Si no se halla presente, se añade $(t.A, 1)$ a la vista materializada. Si el grupo $t.A$ se halla presente, se añade 1 a su cuenta.

  Cuando un conjunto e_r se elimina de r, para cada tupla t de e_r se hace lo siguiente. Se busca el grupo $t.A$ en la vista materializada y se resta 1 de la cuenta del grupo. Si la cuenta se hace 0, se elimina la tupla para el grupo $t.A$ de la vista materializada.

- **sum**: Considérese una vista materializada $v = {}_A\mathcal{G}_{sum(B)}(r)$.

  Cuando un conjunto de tuplas i_r se inserta en r, para cada tupla t de i_r se hace lo siguiente. Se busca el grupo $t.A$ en la vista materializada. Si no se halla presente se añade $(t.A, t.B)$ a la vista materializada; además, se almacena una cuenta de 1 asociada con $(t.A, t.B)$, igual que se hizo para las proyecciones. Si el grupo $t.A$ se halla presente, se añade el valor de $t.B$ al valor agregado para el grupo y se añade 1 a la cuenta del grupo.

  Cuando se elimina un conjunto de tuplas *er* de r, para cada tupla t de *er* hay que hacer lo siguiente. Se busca el grupo $t.A$ en la vista materializada y se resta $t.B$ del valor agregado para el grupo. También se resta 1 de la cuenta del grupo y,

si la cuenta llega a 0, se elimina la tupla para el grupo *t.A* de la vista materializada.

Si no se guardara el valor de cuenta adicional no se podría distinguir el caso de que la suma para el grupo sea 0 del caso en que se ha eliminado la última tupla de un grupo.

- **avg**: Considérese una vista materializada $v = {}_A\mathcal{G}_{avg(B)}(r)$.

  La actualización directa del promedio de una inserción o de una eliminación no resulta posible, ya que no sólo depende del promedio antiguo y de la tupla que se inserta o elimina, sin también del número de tuplas del grupo.

  En lugar de eso, para tratar el caso de **avg**, se conservan los valores de agregación **sum** y **count** como se describieron anteriormente y se calcula el promedio como la suma dividida por la cuenta.

- **min, max**: Considérese una vista materializada $v = {}_A\mathcal{G}_{min(B)}(r)$. (El caso de **max** es completamente equivalente.)

  El tratamiento de las inserciones en *r* es inmediato. La conservación de los valores de agregación **min** y **max** para las eliminaciones puede resultar más costoso. Por ejemplo, si la tupla correspondiente al valor mínimo para un grupo se elimina de *r*, hay que examinar las demás tuplas de *r* que están en el mismo grupo para hallar el nuevo valor mínimo.

14.5.2.4. Otras operaciones

La operación de conjuntos *intersección* se conserva de la manera siguiente. Dada la vista materializada $v = r \cap s$, cuando una tupla se inserta en *r* se comprueba si está presente en *s* y, en caso afirmativo, se añade a *v*. Si se elimina una tupla de *r*, se elimina de la intersección si se halla presente.

Las otras operaciones con conjuntos, *unión* y *diferencia de conjuntos*, se tratan de manera parecida; los detalles se dejan al lector.

Las reuniones externas se tratan de manera muy parecida a las reuniones, pero con algún trabajo adicional. En el caso de la eliminación de *r* hay que manejar las tuplas de *s* que ya no coinciden con ninguna tupla de *r*. En el caso de una inserción en *r*, hay que manejar las tuplas de *s* que no coincidían con ninguna tupla de *r*. De nuevo se dejan los detalles al lector.

14.5.2.5. Tratamiento de expresiones

Hasta ahora se ha visto el modo de actualizar de manera incremental el resultado de una sola operación. Para tratar una expresión entera se pueden obtener expresiones para el cálculo del cambio incremental en el resultado de cada subexpresión, comenzando por las de menor tamaño.

Por ejemplo, supóngase que se desea actualizar de manera incremental la vista materializada $E_1 \bowtie E_2$ cuando se inserta un conjunto de tuplas i_r en la relación *r*. Supóngase que *r* se utiliza sólo en E_1. Supóngase que el conjunto de tuplas que se va a insertar en E_1 viene dado por la expresión D_1. Entonces, la expresión $D_1 \bowtie E_2$ da el conjunto de tuplas que hay que insertar en $E_1 \bowtie E_2$.

Véanse las notas bibliográficas para obtener más detalles sobre la conservación incremental de las vistas con expresiones.

14.5.3. Optimización de consultas y vistas materializadas

La optimización de consultas puede llevarse a cabo tratando las vistas materializadas igual que a las relaciones normales. No obstante, las vistas materializadas ofrecen más oportunidades para la optimización:

- Reescritura de las consultas para el empleo de vistas materializadas:

  Supóngase que está disponible la vista materializada $v = r \bowtie s$ y que un usuario emite la consulta $r \bowtie s \bowtie t$. Puede que la reescritura de la consulta como $v \bowtie t$ proporcione un plan de consulta más eficiente que la optimización de la consulta tal y como se ha emitido. Por tanto, es función del optimizador de consultas reconocer si se puede utilizar una vista materializada para acelerar una consulta.

- Sustitución del empleo de una vista materializada por la definición de la vista:

  Supóngase que está disponible la vista materializada $v = r \bowtie s$, pero sin ningún índice definido sobre ella, y que un usuario emite la consulta $\sigma_{A=10}(v)$. Supóngase también que *s* tiene un índice sobre el atributo común *B*, y que *r* tiene un índice sobre el atributo *A*. Puede que el mejor plan para esta consulta sea sustituir *v* por $r \bowtie s$, lo que puede llevar al plan de consulta $\sigma_{A=10}(r) \bowtie s$; la selección y la reunión pueden llevarse a cabo de manera eficiente empleando los índices sobre *r.A* y sobre *s.B*, respectivamente. Por el contrario, puede que la evaluación de la selección directamente sobre *v* necesite una exploración completa de *v*, lo que puede resultar más costoso.

Las notas bibliográficas dan indicaciones para investigar el modo de llevar a cabo de manera eficiente la optimización de las consultas con vistas materializadas.

Otro problema de optimización relacionado es el de la **selección de las vistas materializadas**, es decir, la identificación del mejor conjunto de vistas para su materialización. Esta decisión debe tomarse con base en la **carga de trabajo** del sistema, que es una secuencia de consultas y de actualizaciones que refleja la carga típica del sistema. Un criterio sencillo sería la selección de un conjunto de vistas materializadas que minimice el tiempo global de ejecución de la carga de trabajo de consultas y de actualizaciones, incluido el tiempo empleado para conservar las vistas materializadas. Los administradores de bases de datos suelen modificar este criterio para tener en

cuenta la importancia de las diferentes consultas y actualizaciones: puede ser necesaria una respuesta rápida para algunas consultas y actualizaciones, mientras que puede resultar aceptable una respuesta lenta para otras. Los índices son como las vistas materializadas, en el sentido de que también son datos obtenidos, pueden acelerar las consultas y pueden desacelerar las actualizaciones. Por tanto, el problema de la **selección de índices** se halla íntimamente relacionado con el de la selección de las vistas materializadas, aunque resulta más sencillo.

Estos problemas se examinan con más detalle en los apartados 21.2.5 y 21.2.6.

Algunos sistemas de bases de datos, como SQL Server 7.5 de Microsoft y RedBrick Data Warehouse de Informix, proporcionan herramientas para ayudar a los administradores de bases de datos en la selección de los índices y de las vistas materializadas. Estas herramientas examinan el historial de consultas y de actualizaciones y sugieren los índices y las vistas que hay que materializar.

14.6. RESUMEN

- Dada una consulta, suele haber gran variedad de métodos para calcular la respuesta. Es responsabilidad del sistema transformar la consulta tal y como la introdujo el usuario en una consulta equivalente que pueda calcularse de manera más eficiente. El proceso de búsqueda de una buena estrategia para el procesamiento de la consulta se denomina *optimización de consultas*.
- La evaluación de las consultas complejas implica muchos accesos a disco. Dado que la transferencia de los datos desde el disco resulta lenta en comparación con la velocidad de la memoria principal y de la CPU del sistema informático, merece la pena asignar una cantidad considerable de procesamiento a la elección de un método que minimice los accesos al disco.
- La estrategia que escoja el sistema de bases de datos para la evaluación de una operación depende del tamaño de cada relación y de la distribución de los valores dentro de las columnas. Para que puedan basar su elección de estrategia en información de confianza, los sistemas de bases de datos almacenan estadísticas para cada relación *r*. Entre estas estadísticas están
  - El número de tuplas de la relación *r*
  - El tamaño del registro (tupla) de la relación *r* en bytes
  - El número de valores diferentes que aparecen en la relación *r* para un atributo determinado
- Estas estadísticas permiten estimar el tamaño del resultado de varias operaciones, así como el coste de su ejecución. La información estadística sobre las relaciones resulta especialmente útil cuando se dispone de varios índices para ayudar al procesamiento de una consulta. La presencia de estas estructuras tiene una influencia significativa en la elección de una estrategia de procesamiento de consultas.
- Cada expresión del álgebra relacional representa una secuencia concreta de operaciones. El primer paso para la selección de una estrategia de procesamiento de consultas es la búsqueda de una expresión del álgebra relacional que sea equivalente a la expresión dada y que se estime menos costosa de ejecutar.
- Hay varias reglas de equivalencia que se pueden emplear para transformar una expresión en otra equivalente. Estas reglas se emplean para generar de manera sistemática todas las expresiones equivalentes a la consulta dada.
- Los planes de evaluación alternativa para cada expresión pueden generarse mediante reglas parecidas y se puede escoger el plan más económico para todas las expresiones. Se dispone de varias técnicas de optimización para reducir el número de expresiones alternativas y los planes que hace falta generar.
- La heurística se emplea para reducir el número de planes considerados y, por tanto, para reducir el coste de la optimización. Entre las reglas heurísticas para transformar las consultas del álgebra relacional están «Llevar a cabo las operaciones de selección tan pronto como sea posible», «Llevar a cabo las proyecciones tan pronto como sea posible» y «Evitar los productos cartesianos».
- Las vistas materializadas pueden utilizarse para acelerar el procesamiento de las consultas. La conservación incremental de las vistas es necesaria para actualizar de forma eficiente las vistas materializadas cuando se modifican las relaciones subyacentes. El diferencial de cada operación puede calcularse mediante expresiones algebraicas que impliquen a los diferenciales de las entradas de la operación. Entre otros aspectos relacionados con las vistas materializadas están el modo de optimizar las consultas haciendo uso de las vistas materializadas disponibles y el modo de seleccionar las vistas que hay que materializar.

TÉRMINOS DE REPASO

- Conjunto mínimo de reglas de equivalencia
- Conservación de las vistas materializadas
  - Recálculo
  - Mantenimiento incremental
  - Inserción
  - Eliminación
  - Actualización
- Descorrelación
- Enumeración de las expresiones equivalentes
- Equivalencia de las expresiones
- Estimación de la estadística
- Estimación del tamaño
  - Selección
  - Selectividad
  - Reunión
- Estimación de los valores distintos
- Evaluación correlacionada
- Información de catálogo
- Interacción de las técnicas de evaluación
- Optimización basada en el coste
- Optimización de consultas
- Optimización de las consultas con las vistas materializadas
- Optimización heurística
- Optimización del orden de reunión
  - Algoritmo de programación dinámica
  - Orden de reunión en profundidad por la izquierda
- Reglas de equivalencia
  - Conmutatividad de la reunión
  - Asociatividad de la reunión
- Selección de índices
- Selección del plan de acceso
- Selección de los planes de evaluación
- Selección de las vistas materializadas
- Transformación de las expresiones
- Vistas materializadas

EJERCICIOS

14.1. El agrupamiento de los índices puede permitir un acceso más rápido a los datos de lo que permiten los índices no agrupados. Indíquese el momento en que se deben crear índices no agrupados pese a las ventajas de los índices agrupados. Explíquese la respuesta.

14.2. Considérense las relaciones r_1 (*A, B, C*), r_2 (*C, D, E*) y r_3 (*E, F*), con las claves principales *A*, *C* y *E*, respectivamente. Supóngase que r_1 tiene 1.000 tuplas, r_2 tiene 1.500 tuplas y r_3 tiene 750 tuplas. Estímese el tamaño de $r_1 \bowtie r_2 \bowtie r_3$ y diséñese una estrategia eficiente para el cálculo de la reunión.

14.3. Considérense las relaciones r_1 (*A, B, C*), r_2 (*C, D, E*) y r_3 (*E, F*) del Ejercicio 14.2. Supóngase que no hay claves principales, excepto el esquema completo. Sean $V(C, r_1)$ 900, $V(C, r_2)$ 1100, $V(E, r_2)$ 50 y $V(E, r_3)$ 100. Supóngase que r_1 tiene 1.000 tuplas, r_2 tiene 1.500 tuplas y r_3 tiene 750 tuplas. Estímese el tamaño de $r_1 \bowtie r_2 \bowtie r_3$ y diséñese una estrategia eficiente para calcular la reunión.

14.4. Supóngase que se dispone de un árbol B+ para *ciudad-sucursal* para la relación *sucursal* y que no se dispone de ningún otro índice. Indíquese la mejor manera de tratar las siguientes selecciones que implican a la negación.

a. $\sigma_{\neg(\textit{ciudad-sucursal} < \text{«Arganzuela»})}(\textit{sucursal})$
b. $\sigma_{\neg(\textit{ciudad-sucursal} = \text{«Arganzuela»})}(\textit{sucursal})$
c. $\sigma_{\neg(\textit{ciudad-sucursal} < \text{«Arganzuela»} \vee \textit{activos} < 5000)}(\textit{sucursal})$

14.5. Supóngase que se dispone de un árbol B+ para (*nombre-sucursal, ciudad-sucursal*) para la relación *sucursal*. Indíquese la mejor manera de tratar la selección siguiente.

$$\sigma_{(\textit{ciudad-sucursal} < \text{«Arganzuela»}) \wedge (\textit{activos} < 5000) \wedge (\textit{nombre-sucursal} = \text{«Centro»})}(\textit{sucursal})$$

14.6. Demuéstrese que se cumplen las equivalencias siguientes. Explíquese el modo en que se pueden aplicar para mejorar la eficiencia de determinadas consultas:

a. $E_1 \bowtie_\theta (E_2 - E_3) = (E_1 \bowtie_\theta E_2 - E_1 \bowtie_\theta E_3)$.
b. $\sigma_\theta ({}_A\mathcal{G}_F (E)) = {}_A\mathcal{G}_F (\sigma_\theta (E))$, donde θ sólo utiliza atributos de *A*.
c. $\sigma_\theta (E_1 ⟖ E_2) = \sigma_\theta (E_1) ⟖ E_2$, donde θ sólo utiliza atributos de E_1.

14.7. Muéstrese el modo de obtener las equivalencias siguientes mediante una secuencia de transformaciones utilizando las reglas de equivalencia del Apartado 14.3.1.

a. $\sigma_{\theta_1 \wedge \theta_2 \wedge \theta_3}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(E)))$
b. $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta_3} E_2) = \sigma_{\theta_1}(E_1 \bowtie_{\theta_3} (\sigma_{\theta_2}(E_2)))$, donde θ_2 sólo implica atributos de E_2

14.8. Para cada uno de los siguientes pares de expresiones, dense ejemplos de relaciones que muestren que las expresiones no son equivalentes.

a. $\Pi_A (R - S)$ y $\Pi_A (R) - \Pi_A (S)$

b. $\sigma_{B<4} ({}_A\mathcal{G}_{max(B)} (R))$ y ${}_A\mathcal{G}_{max(B)} (\sigma_{B<4} (R))$

c. En las expresiones anteriores, si las dos apariciones de *max* se sustituyeran por *min,* indicar si las expresiones serían equivalentes.

d. $(R ⟗ S) ⟗ T$ y $R ⟗ (S ⟗ T)$
En otras palabras, la reunión externa por la izquierda no es asociativa.
(Sugerencia: Supóngase que los esquemas de las tres relaciones son $R(a, b1)$, $S(a, b2)$ y $T(a, b3)$, respectivamente).

e. $\sigma_\theta (E_1 ⟗ E_2)$ y $E_1 ⟗ \sigma_\theta (E_2)$, donde θ utiliza sólo atributos de E_2

14.9. SQL permite las relaciones con duplicados (Capítulo 4).

a. Defínanse las versiones de las operaciones básicas del álgebra relacional $\sigma, \Pi, \times, \bowtie, -, \cup$ y $\cap$ que trabajan en relaciones con duplicados, de manera consistente con SQL.

b. Compruébese cuáles de las reglas de equivalencia de la 1 a la 7.b se cumplen para la versión multiconjunto del álgebra relacional definida en el apartado a.

14.10. ** Demuéstrese que, con n relaciones, hay $(2(n - 1))!/(n - 1)!$ órdenes de reunión diferentes.

Sugerencia: Un **árbol binario completo** es aquel en el que cada nodo interno tiene exactamente dos hijos. Utilícese el hecho de que el número de árboles binarios completos diferentes con n nodos hojas es $\frac{1}{n}\binom{2(n-1)}{(n-1)}$.

Si se desea, se puede obtener la fórmula para el número de árboles binarios completos con n nodos a partir de la fórmula para el número de árboles binarios con n nodos. El número de árboles binarios con n nodos es $\frac{1}{n+1}\binom{2n}{n}$; este número se conoce como número de Catalan y su obtención puede hallarse en cualquier libro de texto estándar sobre estructuras de datos o algoritmos.

14.11. ** Demuéstrese que el orden de reunión de menor coste puede calcularse en un tiempo $O(3^n)$. Supóngse que se puede almacenar y examinar la información sobre un conjunto de relaciones (como el orden óptimo de reunión para el conjunto y el coste de ese orden de reunión) en un tiempo constante. (Si se encuentra difícil este ejercicio, demuéstrese al menos la cota de tiempo menos estricta de $O(2^{2n})$.)

14.12. Demuéstrese que, si sólo se toman en consideración los árboles de reunión en profundidad por la izquierda, como en el optimizador System R, el tiempo empleado en buscar el orden de reunión más eficiente es del orden de $n2^n$. Supóngase que sólo hay un orden interesante.

14.13. Se dice que un conjunto de reglas de equivalencia está *completo* si, siempre que dos expresiones son equivalentes, se puede obtener una de la otra mediante una secuencia de utilizaciones de las reglas de equivalencia. Indíquese si el conjunto de reglas de equivalencia que se consideró en el Apartado 14.3.1 es completo. Sugerencia: considérese la equivalencia $\sigma_{3=5} (r) = \{ \}$.

14.14. Descorrelación:

a. Escríbase una consulta anidada sobre la relación *cuenta* para buscar para cada sucursal cuyo nombre comience por «B» todas las cuentas con el saldo máximo de cada sucursal.

b. Reescríbase la consulta anterior, sin emplear consultas anidadas; en otras palabras, descorrelaciónese la consulta.

c. Diséñese un procedimiento (parecido al descrito en el Apartado 14.4.5) para descorrelacionar estas consultas.

14.15. Descríbase el modo de conservar de manera incremental el resultado de las operaciones siguientes, tanto para inserciones como para eliminaciones.

a. Unión y diferencia de conjuntos

b. Reunión externa por la izquierda

14.16. Dese un ejemplo de expresión que defina una vista materializada y dos situaciones (conjuntos de estadísticas para las relaciones de entrada y sus diferenciales) tales que la conservación incremental de la vista sea mejor que su recálculo en una de las situaciones y el recálculo sea mejor en la otra.

NOTAS BIBLIOGRÁFICAS

El trabajo precursor de Selinger et al. [1979] describe la selección del camino de acceso en el optimizador System R, que fue uno de los primeros optimizadores de consultas relacionales. Graefe y McKenna [1993] describen Volcano, un optimizador de consultas basado en reglas de equivalencia. El procesamiento de consultas en Starburst se describe en Haas et al. [1989]. La optimización de consultas en Oracle se describe brevemente en Oracle [1997].

La estimación de las estadísticas de los resultados de las consultas, como el tamaño del resultado, se aborda en Ioannidis y Poosala [1995], Poosala et al. [1996] y Ganguly et al. [1996], entre otros. Las distribuciones no uniformes de valores causan problemas para la estimación del tamaño y del coste de las consultas. Las técnicas de estimación del coste que utilizan histogramas de las distribuciones de los valores se han propuesto para abordar el problema. Ioannidis y Christodoulakis [1993], Ioannidis y Poosala [1995] y Poosala et al. [1996] presentan los resultado en esta área.

La búsqueda exhaustiva de todos los planes de consultas no resulta práctica para la optimizacion de las reu-

niones que implican a muchas relaciones y se han propuesto técnicas basadas en la búsqueda aleatoria, que no examinan todas las alternativas. Ioannidis y Wong [1987], Swami y Gupta [1988] y Ioannidis y Kang [1990] presentan resultados en esta área.

Ioannidis et al. [1992] y Ganguly [1998] han propuesto técnicas de *optimización paramétrica de consultas* para tratar el procesamiento de las consultas cuando la selectividad de los parámetros de la consulta no se conoce en el momento de la optimización. Se calcula un conjunto de planes —uno por cada una de las diferentes selectividades de las consultas—, y el optimizador lo almacena, en el momento de la compilación. Uno de estos planes se elige en el momento de la ejecución, con base en las selectividades reales, evitando el coste de la optimización completa en el momento de la ejecución.

Klug [1982] realizó uno de los primeros trabajos sobre optimización de expresiones del álgebra relacional con funciones de agregación. Entre el trabajo más reciente en esta área están Yan y Larson [1995] y Chaudhuri y Shim [1994]. La optimización de las consultas que contienen reuniones externas se describe en Rosenthal y Reiner [1984], Galindo-Legaria y Rosenthal [1992] y Galindo-Legaria [1994].

El lenguaje SQL plantea varios desafíos para la optimización de las consultas, incluidas la presencia de valores duplicados y nulos y la semántica de las subconsultas anidadas. La extensión del álgebra relacional a los valores duplicados se describe en Dayal et al. [1982]. La optimización de las subconsultas anidadas se trata en Kim [1982], Ganski y Wong [1987], Dayal [1987] y, más recientemente, en Seshadri et al. [1996].

Cuando las consultas se generan mediante vistas, se suelen reunir más relaciones de las necesarias para el cálculo de cada consulta. Se ha agrupado un conjunto de técnicas para la minimización de las reuniones bajo el nombre de *optimización con tableau*. El concepto de *tableau* lo introdujeron Aho et al. [1979b] y Aho et al. [1979a] y lo ampliaron Sagiv y Yannakakis [1981]. Ullman [1988] y Maier [1983] proporcionan un tratamiento de *tableau* con un nivel de libro de texto.

Sellis [1988] y Roy et al. [2000] describen la *optimización multiconsulta*, que es el problema de optimización de la ejecución de varias consultas como si fueran un grupo. Si se toma en consideración todo un grupo de consultas, resulta posible descubrir *subexpresiones comunes* que pueden evaluarse una sola vez para todo el grupo. Finkelstein [1982] y Hall [1976] consideran la optimización de un grupo de consultas y el empleo de las subexpresiones comunes. Dalvi et al. [2001] estudian los problemas de optimización en los encauzamientos con espacio de memorias intermedias limitado combinadas con el compartimiento de subexpresiones comunes.

La optimización de consultas puede hacer uso de la información semántica, como las dependencias funcionales y otras restricciones de integridad. La *optimización semántica de las consultas* en las bases de datos relacionales se estudia en King [1981], Chakravarthy et al. [1990] y, en el contexto de la agregación, en Sudarshan y Ramakrishnan [1991].

Las técnicas de procesamiento y de optimización de consultas para Datalog, en especial las técnicas para tratar las consultas sobre las vistas recursivas se describen en Bancilhon y Ramakrishnan [1986], Beeri y Ramakrishnan [1991], Ramakrishnan et al. [1992c], Srivastava et al. [1995] y Mumick et al. [1996]. Las técnicas de procesamiento y de optimización de consultas para las bases de datos orientadas a los objetos se estudian en Maier y Stein [1986], Beech [1988], Bertino y Kim [1989] y Blakeley et al. [1993].

Blakeley et al. [1986], Blakeley et al. [1989] y Griffin y Libkin [1995] describen las técnicas para la conservación de las vistas materializadas. Gupta y Mumick [1995] proporcionan una reseña de la conservación de las vistas materializadas. La optimización de los planes de conservación de las vistas materializadas se describe en Vista [1998] y Mistry et al. [2001]. La optimización de consultas en presencia de vistas materializadas se aborda en Larson y Yang [1985], Chaudhuri et al. [1995], Dar et al. [1996] y Roy et al. [2000]. La selección de índices y la selección de vistas materializadas se aborda en Ross et al. [1996], Labio et al. [1997], Gupta [1997], Chaudhuri y Narasayya [1997] y Roy et al. [2000].

PARTE

V

GESTIÓN DE TRANSACCIONES

El término transacción hace referencia a un conjunto de operaciones que forman una única unidad lógica de trabajo. Por ejemplo, la transferencia de dinero de una cuenta a otra es una transacción que consta de dos actualizaciones, una para cada cuenta.

Resulta importante que, o bien se ejecuten completamente todas las acciones de una transacción, o bien, en caso de fallo, se deshagan los efectos parciales de la transacción. Esta propiedad se denomina *atomicidad*. Además, una vez ejecutada con éxito una transacción, sus efectos deben persistir en la base de datos: un fallo en el sistema no debe tener como consecuencia que la base de datos se olvide de una transacción que haya completado con éxito. Esta propiedad se denomina *durabilidad*.

En los sistemas de bases de datos en los que se ejecutan de manera concurrente varias transacciones, si no se controlan las actualizaciones de los datos compartidos existe la posibilidad de que las transacciones vean estados intermedios inconsistentes creados por las actualizaciones de otras transacciones. Esta situación puede dar lugar a actualizaciones erróneas de los datos almacenados en la base de datos. Por tanto, los sistemas de bases de datos deben proporcionar los mecanismos para aislar las transacciones de otras transacciones que se ejecuten de manera concurrente. Esta propiedad se denomina *aislamiento*.

El Capítulo 15 describe con detalle el concepto de transacción, incluidas las propiedades de atomicidad, durabilidad, aislamiento y otras propiedades proporcionadas por la abstracción de las transacciones. En concreto, el capítulo precisa el concepto de aislamiento por medio de un concepto denominado secuencialidad.

El Capítulo 16 describe varias técnicas de control de la concurrencia que ayudan a implementar la propiedad del aislamiento.

El Capítulo 17 describe el componente de las bases de datos para la administración de las recuperaciones, que implementa las propiedades de atomicidad y de durabilidad.

CAPÍTULO

15 TRANSACCIONES

A menudo, desde el punto de vista del usuario de una base de datos, se considera a un conjunto de varias operaciones sobre una base de datos como una única operación. Por ejemplo, una transferencia de fondos desde una cuenta corriente a una cuenta de ahorros es una operación simple desde el punto de vista del cliente; sin embargo, en el sistema de base de datos, está compuesta internamente por varias operaciones. Evidentemente es esencial que tengan lugar todas las operaciones o que, en caso de fallo, ninguna de ellas se produzca. Sería inaceptable efectuar el cargo de la transferencia en la cuenta corriente y que no se abonase en la cuenta de ahorros.

Se llama **transacción** a una colección de operaciones que forman una única unidad lógica de trabajo. Un sistema de base de datos debe asegurar que la ejecución de las transacciones se realice adecuadamente a pesar de la existencia de fallos: o se ejecuta la transacción completa o no se ejecuta en absoluto. Además debe gestionar la ejecución concurrente de las transacciones evitando introducir inconsistencias. Volviendo al ejemplo de la transferencia de fondos, una transacción que calcule el saldo total del cliente podría ver el saldo de la cuenta corriente antes de que sea cargado por la transacción de la transferencia de fondos, y el saldo de la cuenta de ahorros después del abono. Como resultado, se obtendría un resultado incorrecto.

Este capítulo es una introducción a los conceptos básicos en el procesamiento de transacciones. En los Capítulos 16 y 17 se incluyen más detalles sobre el procesamiento concurrente de transacciones y la recuperación de fallos. En el Capítulo 24 se tratan temas adicionales acerca del procesamiento de transacciones.

15.1. CONCEPTO DE TRANSACCIÓN

Una **transacción** es una **unidad** de la ejecución de un programa que accede y posiblemente actualiza varios elementos de datos. Una transacción se inicia por la ejecución de un programa de usuario escrito en un lenguaje de manipulación de datos de alto nivel o en un lenguaje de programación (por ejemplo SQL, COBOL, C, C++ o Java), y está delimitado por instrucciones (o llamadas a función) de la forma **inicio transacción** y **fin transacción**. La transacción consiste en todas las operaciones que se ejecutan entre **inicio transacción** y el **fin transacción**.

Para asegurar la integridad de los datos se necesita que el sistema de base de datos mantenga las siguientes propiedades de las transacciones:

- **Atomicidad.** O todas las operaciones de la transacción se realizan adecuadamente en la base de datos o ninguna de ellas.
- **Consistencia.** La ejecución aislada de la transacción (es decir, sin otra transacción que se ejecute concurrentemente) conserva la consistencia de la base de datos.
- **Aislamiento.** Aunque se ejecuten varias transacciones concurrentemente, el sistema garantiza que para cada par de transacciones T_i y T_j, se cumple que para los efectos de T_i, o bien T_j ha terminado su ejecución antes de que comience T_i, o bien que T_j ha comenzado su ejecución después de que T_i termine. De este modo, cada transacción ignora al resto de las transacciones que se ejecuten concurrentemente en el sistema.
- **Durabilidad.** Tras la finalización con éxito de una transacción, los cambios realizados en la base de datos permanecen, incluso si hay fallos en el sistema.

Estas propiedades a menudo reciben el nombre de **propiedades ACID**; el acrónimo se obtiene de la primera letra de cada una de las cuatro propiedades en inglés (*Atomicity, Consistency, Isolation* y *Durability*, respectivamente).

Para comprender mejor las propiedades ACID y la necesidad de dichas propiedades, considérese un sistema bancario simplificado constituido por varias cuentas y un conjunto de transacciones que acceden y actualizan dichas cuentas. Por ahora se asume que la base de datos reside permanentemente en disco, pero una porción de la misma reside temporalmente en la memoria principal.

El acceso a la base de datos se lleva a cabo mediante las dos operaciones siguientes:

- leer(X), que transfiere el dato X de la base de datos a una memoria intermedia local perteneciente a la transacción que ejecuta la operación leer.
- escribir(X), que transfiere el dato X desde la memoria intermedia local de la transacción que ejecuta la operación escribir a la base de datos.

En un sistema de base de datos real, la operación escribir no tiene por qué producir necesariamente una actualización de los datos en disco; la operación escribir puede almacenarse temporalmente en memoria y llevarse a disco más tarde. Sin embargo, por el momento se supondrá que la operación escribir actualiza inmediatamente la base de datos. Se volverá a este tema en el Capítulo 17.

Sea T_i una transacción para transferir 50 € de la cuenta A a la cuenta B. Se puede definir dicha transacción como

T_i: leer(A);
$A := A - 50$;
escribir(A);
leer(B);
$B := B + 50$;
escribir(B).

Considérense ahora cada uno de los requisitos ACID (para una presentación más cómoda, se consideran en distinto orden al indicado por A-C-I-D).

- **Consistencia:** en este caso el requisito de consistencia es que la suma de A y B no sea alterada al ejecutar la transacción. Sin el requisito de consistencia, ¡la transacción podría crear o destruir dinero! Se puede comprobar fácilmente que si una base de datos es consistente antes de ejecutar una transacción, sigue siéndolo después de ejecutar dicha transacción.

  La responsabilidad de asegurar la consistencia de una transacción es del programador de la aplicación que codifica dicha transacción. La comprobación automática de las restricciones de integridad puede facilitar esta tarea, como se vio en el Capítulo 6.

- **Atomicidad:** supóngase que justo antes de ejecutar la transacción T_i los valores de las cuentas A y B son de 1.000 € y de 2.000 €, respectivamente. Supóngase ahora que durante la ejecución de la transacción T_i se produce un fallo que impide que dicha transacción finalice con éxito su ejecución. Ejemplos de este tipo de fallos pueden ser los fallos en la alimentación, los fallos del hardware y los errores software. Además, supóngase que el fallo tiene lugar después de ejecutarse la operación escribir(A), pero antes de ejecutarse la operación escribir(B). Es ese caso, los valores de las cuentas A y B que se ven reflejados en la base de datos son 950 € y 2.000 €. Se han perdido 50 € de la cuenta A como resultado de este fallo. En particular se puede ver que ya no se conserva la suma $A + B$.

  Así, como resultado del fallo, el estado del sistema deja de reflejar el estado real del mundo que se supone que modela la base de datos. Un estado así se denomina **estado inconsistente**. Hay que asegurarse de que estas inconsistencias no sean visibles en un sistema de base de datos. Nótese, sin embargo, que un sistema puede en algún momento alcanzar un estado inconsistente. Incluso si la transacción T_i se ejecuta por completo, existe un punto en el que el valor de la cuenta A es de 950 € y el de la cuenta B es de 2.000 €, lo cual constituye claramente un estado inconsistente. Este estado, sin embargo, se sustituye eventualmente por otro estado consistente en el que el valor de la cuenta A es de 950 € y el de la cuenta B es de 2.050 €. De este modo, si la transacción no empieza nunca o se garantiza que se complete, un estado inconsistente así no será visible excepto durante la ejecución de la transacción. Ésta es la razón de que aparezca el requisito de atomicidad. Si se proporciona la propiedad de atomicidad, o todas las acciones de la transacción se ven reflejadas en la base de datos, o ninguna de ellas.

  La idea básica que hay detrás de asegurar la atomicidad es la siguiente. El sistema de base de datos mantiene los valores antiguos (en disco) de aquellos datos sobre los que una transacción realiza una escritura y, si la transacción no completa su ejecución, los valores antiguos se recuperan para que parezca que la transacción no se ha ejecutado. Estas ideas se muestran más adelante en el Apartado 15.2. La responsabilidad de asegurar la atomicidad es del sistema de base de datos; en concreto, lo maneja un componente llamado **componente de gestión de transacciones**, que se describe en detalle en el Capítulo 17.

- **Durabilidad:** una vez que se completa con éxito la ejecución de una transacción, y después de comunicar al usuario que inició la transacción que se ha realizado la transferencia de fondos, no debe suceder que un fallo en el sistema produzca la pérdida de datos correspondientes a dicha transferencia.

  La propiedad de durabilidad asegura que, una vez que se completa con éxito una transacción, persisten todas las modificaciones realizadas en la base de datos, incluso si hay un fallo en el sistema después de completarse la ejecución de dicha transacción.

  A partir de ahora se asume que un fallo en la computadora del sistema produce una pérdida de datos de la memoria principal, pero los datos almacenados en disco nunca se pierden. Se puede garantizar la durabilidad si se asegura que

  1. Las modificaciones realizadas por la transacción se guardan en disco antes de que finalice la transacción.

2. La información de las modificaciones realizadas por la transacción guardada en disco es suficiente para permitir a la base de datos reconstruir dichas modificaciones cuando el sistema se reinicie después del fallo.

La responsabilidad de asegurar la durabilidad es de un componente del sistema de base de datos llamado **componente de gestión de recuperaciones**. El componente de gestión de transacciones y el componente de gestión de recuperaciones están estrechamente relacionados, y su implementación se describe en el Capítulo 17.

- **Aislamiento:** incluso si se aseguran las propiedades de consistencia y de atomicidad para cada transacción, si varias transacciones se ejecutan concurrentemente, se pueden entrelazar sus operaciones de un modo no deseado, produciendo un estado inconsistente.

  Por ejemplo, como se ha visto antes, la base de datos es inconsistente temporalmente durante la ejecución de la transacción para transferir fondos de la cuenta A a la cuenta B, con el total deducido escrito ya en A y el total incrementado todavía sin escribir en B. Si una segunda transacción que se ejecuta concurrente lee A y B en este punto intermedio y calcula $A + B$, observará un valor inconsistente. Además, si esta segunda transacción realiza después modificaciones en A y B basándose en los valores leídos, la base de datos puede permanecer en un estado inconsistente aunque ambas transacciones terminen.

  Una solución para el problema de ejecutar transacciones concurrentemente es ejecutarlas secuencialmente, es decir, una tras otra. Sin embargo, la ejecución concurrente de transacciones produce notables beneficios en el rendimiento, como se verá en el Apartado 15.4. Se han desarrollado otras soluciones que permiten la ejecución concurrente de varias transacciones.

  Los problemas que causa la ejecución concurrente de transacciones se muestra en el Apartado 15.4. La propiedad de aislamiento asegura que el resultado obtenido al ejecutar concurrentemente las transacciones es un estado del sistema equivalente a uno obtenido al ejecutar una tras otra en algún orden. Los principios de aislamiento se presentarán más adelante en el Apartado 15.5. La responsabilidad de asegurar la propiedad de aislamiento es de un componente del sistema de base de datos llamado **componente de control de concurrencia**, que se presenta en el Capítulo 16.

15.2. ESTADOS DE UNA TRANSACCIÓN

En ausencia de fallos, todas las transacciones se completan con éxito. Sin embargo, como se ha visto antes, una transacción puede que no siempre termine su ejecución con éxito. Una transacción de este tipo se denomina **abortada**. Si se pretende asegurar la propiedad de atomicidad, una transacción abortada no debe tener efecto sobre el estado de la base de datos. Así, cualquier cambio que haya hecho la transacción abortada sobre la base de datos debe deshacerse. Una vez que se han deshecho los cambios efectuados por la transacción abortada, se dice que la transacción se ha **retrocedido**. Parte de la responsabilidad del esquema de recuperaciones es gestionar las transacciones abortadas.

Una transacción que termina con éxito se dice que está **comprometida**. Una transacción comprometida que haya hecho modificaciones transforma la base de datos llevándola a un nuevo estado consistente, que permanece incluso si hay un fallo en el sistema.

Cuando una transacción se ha comprometido no se pueden deshacer sus efectos abortándola. La única forma de deshacer los cambios de una transacción comprometida es ejecutando una **transacción compensadora**. Por ejemplo, si una transacción añade 20 € a una cuenta, la transacción compensadora debería restar 20 € de la cuenta. Sin embargo, no siempre se puede crear dicha transacción compensadora. Por tanto, se deja al usuario la responsabilidad de crear y ejecutar transacciones compensadoras, y no la gestiona el sistema de base de datos. En el Capítulo 24 se incluye un estudio de las transacciones compensadoras.

Es necesario precisar qué se entiende por *terminación con éxito* de una transacción. Se establece por tanto un modelo simple abstracto de transacción. Una transacción debe estar en uno de los estados siguientes:

- **Activa**, el estado inicial; la transacción permanece en este estado durante su ejecución.
- **Parcialmente comprometida**, después de ejecutarse la última instrucción.
- **Fallida**, tras descubrir que no puede continuar la ejecución normal.
- **Abortada**, después de haber retrocedido la transacción y restablecido la base de datos a su estado anterior al comienzo de la transacción.
- **Comprometida**, tras completarse con éxito.

El diagrama de estados correspondiente a una transacción se muestra en la Figura 15.1. Se dice que una transacción se ha comprometido sólo si ha llegado al estado comprometida. Análogamente, se dice que una transacción ha abortado sólo si ha llegado al estado abor-

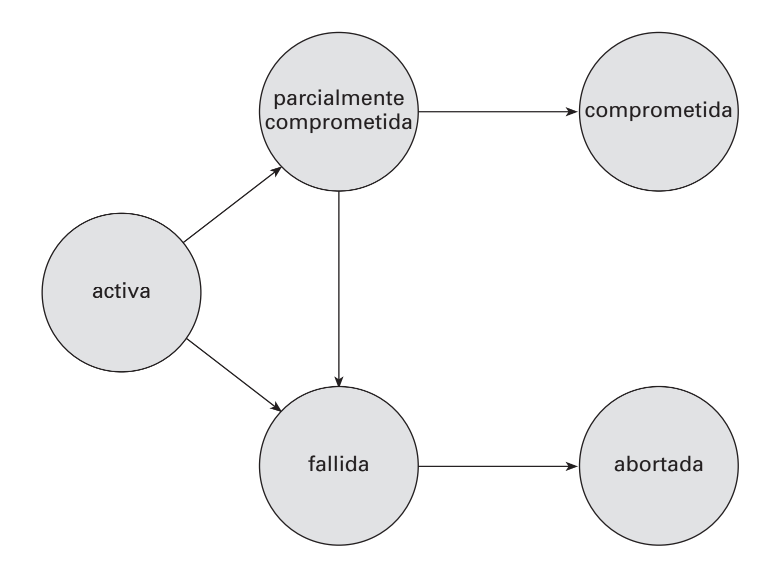


FIGURA 15.1. Diagrama de transición de estado de una transacción.

tada. Una transacción se dice que ha **terminado** si se ha comprometido o bien se ha abortado.

Una transacción comienza en el estado activa. Cuando acaba su última instrucción pasa al estado de parcialmente comprometida. En este punto la transacción ha terminado su ejecución, pero es posible que aún tenga que ser abortada, puesto que los datos actuales pueden estar todavía en la memoria principal y puede producirse un fallo en el hardware antes de que se complete con éxito.

El sistema de base de datos escribe en disco la información suficiente para que, incluso al producirse un fallo, puedan reproducirse los cambios hechos por la transacción al reiniciar el sistema tras el fallo. Cuando se termina de escribir esta información, la transacción pasa al estado comprometida.

Como se ha mencionado antes, se asume que los fallos no provocan pérdidas de datos en disco. Las técnicas para tratar las pérdidas de datos en disco se muestran en el Capítulo 17.

Una transacción llega al estado fallida después de que el sistema determine que dicha transacción no puede continuar su ejecución normal (por ejemplo, a causa de errores de hardware o lógicos). Una transacción de este tipo se debe retroceder. Después pasa al estado abortada. En este punto, el sistema tiene dos opciones:

- **Reiniciar** la transacción, pero sólo si la transacción se ha abortado a causa de algún error hardware o software que no lo haya provocado la lógica interna de la transacción. Una transacción reiniciada se considera una nueva transacción.
- **Cancelar** la transacción. Normalmente se hace esto si hay algún error interno lógico que sólo se puede corregir escribiendo de nuevo el programa de aplicación, o debido a una entrada incorrecta o debido a que no se han encontrado los datos deseados en la base de datos.

Hay que tener cuidado cuando se trabaja con **escrituras externas observables**, como en un terminal o en una impresora. Cuando una escritura así tiene lugar, no puede borrarse puesto que puede haber sido vista fuera del sistema de base de datos. Muchos sistemas permiten que tales escrituras tengan lugar sólo después de que la transacción llegue al estado comprometida. Una manera de implementar dicho esquema es hacer que el sistema de base de datos almacene temporalmente cualquier valor asociado con estas escrituras externas en memoria no volátil, y realice las escrituras actuales sólo si la transacción llega al estado comprometida. Si el sistema falla después de que la transacción llegue al estado comprometida, pero antes de que finalicen las escrituras externas, el sistema de base de datos puede llevar a cabo dichas escrituras externas (usando los datos de la memoria no volátil) cuando el sistema se reinicie.

La gestión de las escrituras externas puede ser más complicada en ciertas situaciones. Por ejemplo, supóngase que la acción externa es dispensar dinero en un cajero automático y el sistema falla justo antes de que se dispense el dinero (se asume que el dinero se dispensa atómicamente). No tiene sentido dispensar el dinero cuando se reinicie el sistema, ya que el usuario probablemente no esté en el cajero. En tal caso es necesario una transacción compensadora, como devolver el dinero a la cuenta del usuario.

Para algunas aplicaciones puede ser deseable permitir a las transacciones activas que muestren datos a los usuarios, particularmente para transacciones de larga duración que se ejecutan durante minutos u horas. Por desgracia no se puede permitir dicha salida de datos observables a no ser que se quiera arriesgar la atomicidad de la transacción. Muchos sistemas de bases de datos actuales aseguran la atomicidad y, por lo tanto, prohíben esta forma de interacción con el usuario. En el Capítulo 24 se describen modelos alternativos que proporcionan transacciones interactivas de larga duración.

15.3. IMPLEMENTACIÓN DE LA ATOMICIDAD Y LA DURABILIDAD

El componente de gestión de recuperaciones de un sistema de base de datos implementa el soporte para la atomicidad y la durabilidad. En primer lugar consideramos un esquema simple pero extremadamente ineficiente, denominado **copia en la sombra**. Este esquema, que se basa en hacer copias de la base de datos, denominadas copias *sombra*, asume que sólo una transacción está activa en cada momento. El esquema asume que la base de datos es simplemente un archivo en disco. En disco se mantiene un puntero llamado puntero_bd que apunta a la copia actual de la base de datos.

En el esquema de copia en la sombra, una transacción que quiera actualizar una base de datos crea primero una copia completa de dicha base de datos. Todos los cambios se hacen en la nueva copia de la base de datos dejando la copia original, la **copia en la sombra**, inalterada. Si en cualquier punto hay que abortar la transacción, la copia nueva simplemente se borra. La copia antigua de la base de datos no se ve afectada.

Si la transacción se completa, se compromete como sigue. En primer lugar se consulta al sistema operativo para asegurarse de que todas las páginas de la nueva copia de la base de datos se han escrito en disco. (En los sistemas Unix se usa la orden fsync para este propósito.) Después de terminar esta orden se actualiza el puntero puntero_bd para que apunte a la nueva copia de la base de datos; la nueva copia se convierte entonces en la copia de la base de datos actual. La copia antigua de la base de datos se borra después. El esquema se describe en la Figura 15.2, en la cual se muestra el estado de la base de datos antes y después de la actualización.

Se dice que la transacción está *comprometida* en el momento en que puntero_bd actualizado se escribe en disco.

Considérese ahora la manera en que esta técnica trata los fallos de las transacciones y del sistema. En primer lugar, considérese un fallo en la transacción. Si la transacción falla en algún momento antes de actualizar puntero_bd, el contenido de la base de datos anterior no se ve afectado. Se puede abortar la transacción simplemente borrando la nueva copia de la base de datos. Una vez que se ha comprometido la transacción, puntero_bd apunta a todas las modificaciones que ésta ha realizado en la base de datos. De este modo, o todas las modificaciones de la transacción se ven reflejadas o ninguna de ellas, independientemente del fallo de la transacción.

Considérese ahora el resultado de un fallo en el sistema. Supóngase que el sistema falla en algún momento antes de escribir en disco el puntero_bd actualizado. Entonces cuando se reinicie el sistema, leerá puntero_bd y verá el contenido original de la base de datos, y ninguno de los efectos de la transacción será visible en la base de datos. A continuación supóngase que el sistema falla después de actualizar en disco puntero_bd. Antes de que el puntero se actualice, todas las páginas actualizadas de la nueva copia de la base de datos se escriben en disco. De nuevo, se asume que una vez que un archivo se escribe en disco, su contenido no se daña incluso si hay un fallo del sistema. Por tanto, cuando el sistema se reinicie, leerá puntero_bd y verá el contenido de la base de datos después que la transacción haya realizado todas las modificaciones.

La implementación depende realmente de que escribir puntero_bd sea una operación atómica; es decir, o se escriben todos sus bytes o ninguno de ellos. Si se escribieran algunos de los bytes del puntero y otros no, el puntero no tendría sentido y al reiniciarse el sistema no se podrían encontrar ni la versión anterior ni la nueva de la base de datos. Afortunadamente los sistemas de disco proporcionan actualizaciones atómicas de bloques enteros, o al menos de un sector del disco. En otras palabras, el sistema de disco garantiza la actualización automática de puntero_bd. siempre que nos aseguremos de

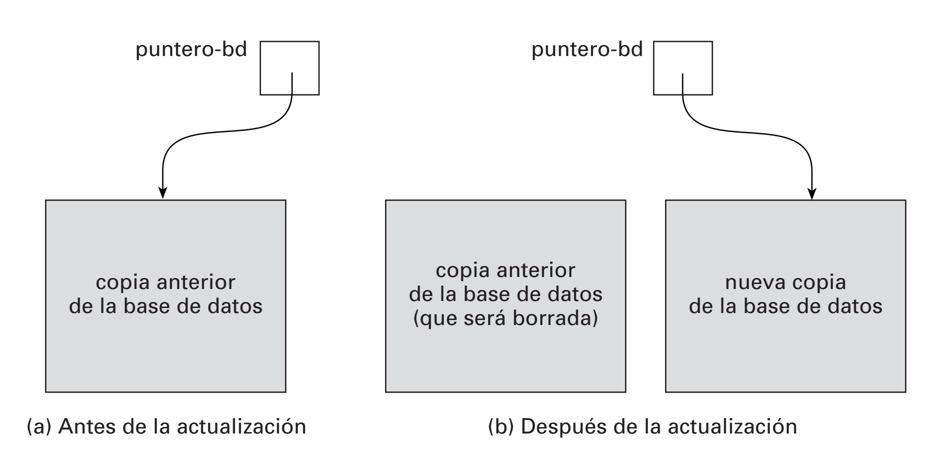


FIGURA 15.2. Técnica de copia en la sombra para la atomicidad y la durabilidad

que puntero_bd cabe en un único sector, lo que se puede asegurar almacenándolo al comienzo de un bloque.

De este modo, la implementación de la copia en la sombra del componente de gestión de recuperaciones asegura las propiedades de atomicidad y durabilidad de las transacciones.

Como ejemplo simple fuera del dominio de las bases de datos, considérese una sesión de edición de texto. Una sesión completa de edición de texto puede modelar una transacción. Las acciones que ejecuta la transacción son leer y actualizar el archivo. Guardar el archivo cuando se termina de editar corresponde a completar la transacción de edición; terminar la sesión de edición sin guardar el archivo corresponde a abortar la transacción de edición.

Muchos editores de texto usan fundamentalmente la implementación que se acaba de describir para asegurar que una sesión de edición es como una transacción. Se usa un nuevo archivo para almacenar las modificaciones. Al finalizar la sesión de edición, si el archivo modificado se va a guardar, se usa una orden para renombrar el nuevo archivo para que tenga el nombre del archivo actual. Se asume que el renombramiento está implementado como una operación atómica en el sistema de archivos, y que al mismo tiempo borra el archivo antiguo.

Por desgracia, esta implementación es extremadamente ineficiente en el contexto de grandes bases de datos, ya que la ejecución de una simple transacción requiere copiar la base de datos *completa*. Además, la implementación no permite a las transacciones ejecutarse concurrentemente unas con otras. Existen formas prácticas de implementar la atomicidad y durabilidad que son mucho menos costosas y más potentes. Estas técnicas se estudian en el Capítulo 17.

15.4. EJECUCIONES CONCURRENTES

Los sistemas de procesamiento de transacciones permiten normalmente la ejecución de varias transacciones concurrentemente. Permitir varias transacciones que actualizan concurrentemente los datos provoca complicaciones en la consistencia de los mismos, como se ha visto antes. Asegurar la consistencia a pesar de la ejecución concurrente de las transacciones requiere un trabajo extra; es mucho más sencillo exigir que las transacciones se ejecuten **secuencialmente**, es decir, una a una, comenzando cada una sólo después de que la anterior se haya completado. Sin embargo, existen dos buenas razones para permitir la concurrencia.

- **Productividad y utilización de recursos mejoradas.** Una transacción consiste en varios pasos. Algunos implican operaciones de E/S; otros implican operaciones de UCP. La UCP y los discos pueden trabajar en paralelo en una computadora. Por tanto, las operaciones de E/S se pueden realizar en paralelo con el procesamiento de la UCP. Se puede entonces explotar el paralelismo de la UCP y del sistema de E/S para ejecutar varias transacciones en paralelo. Mientras una transacción ejecuta una lectura o una escritura en un disco, otra puede ejecutarse en la UCP mientras una tercera transacción ejecuta una lectura o una escritura en otro disco. Todo esto incrementa la **productividad** (*throughput*) del sistema, es decir, en el número de transacciones que puede ejecutar en un tiempo dado. Análogamente, la **utilización** del procesador y del disco aumenta también; en otras palabras, el procesador y el disco están menos tiempo desocupados o sin hacer ningún trabajo útil.
- **Tiempo de espera reducido.** Debe haber una mezcla de transacciones que se ejecutan en el sistema, algunas cortas y otras largas. Si las transacciones se ejecutan secuencialmente, la transacción corta debe esperar a que la transacción larga anterior se complete, lo cual puede llevar a un retardo impredecible en la ejecución de la transacción. Si las transacciones operan en partes diferentes de la base de datos es mejor hacer que se ejecuten concurrentemente, compartiendo los ciclos de la UCP y los accesos a disco entre ambas. La ejecución concurrente reduce los retardos impredecibles en la ejecución de las transacciones. Además se reduce también el **tiempo medio de respuesta**: el tiempo medio desde que una transacción comienza hasta que se completa.

La razón para usar la ejecución concurrente en una base de datos es esencialmente la misma que para usar **multiprogramación** en un sistema operativo.

Cuando se ejecutan varias transacciones concurrentemente, la consistencia de la base de datos se puede destruir a pesar de que cada transacción individual sea correcta. En este apartado se presenta el concepto de planificaciones que ayudan a identificar aquellas ejecuciones que garantizan que se asegura la consistencia.

El sistema de base de datos debe controlar la interacción entre las transacciones concurrentes para evitar que se destruya la consistencia de la base de datos. Esto se lleva a cabo a través de una serie de mecanismos denominados **esquemas de control de concurrencia**. En el Capítulo 16 se estudian los esquemas de control de concurrencia; por ahora nos centraremos en el concepto de ejecución concurrente correcta.

Considérese de nuevo el sistema bancario simplificado del Apartado 15.1, el cual tiene varias cuentas, y un conjunto de transacciones que acceden y modifican dichas cuentas. Sean T_1 y T_2 dos transacciones para

transferir fondos de una cuenta a otra. La transacción T_1 transfiere 50 € de la cuenta A a la cuenta B y se define como sigue

T_1: leer(A);
$A := A - 50$;
escribir(A);
leer(B);
$B := B + 50$;
escribir(B).

La transacción T_2 transfiere el 10 por ciento del saldo de la cuenta A a la cuenta B, y se define

T_2: leer(A);
$temp := A * 0.1$;
$A := A - temp$;
escribir(A);
leer(B);
$B := B + temp$;
escribir(B).

Supóngase que los valores actuales de las cuentas A y B son 1.000 € y 2.000 € respectivamente. Supóngase que las dos transacciones se ejecutan de una en una en el orden T_1 seguida de T_2. Esta secuencia de ejecución se representa en la Figura 15.3. En esta figura la secuencia de pasos o instrucciones aparece en orden cronológico de arriba abajo, con las instrucciones de T_1 en la columna izquierda y las de T_2 en la derecha. Los valores finales de las cuentas A y B, después de que tenga lugar la ejecución de la Figura 15.3, son de 855 € y de 2.145 € respectivamente. De este modo, la suma total de saldo de las cuentas A y B —es decir, la suma $A + B$— se conserva tras la ejecución de ambas transacciones.

Análogamente, si las transacciones se ejecutan de una en una en el orden T_2 seguida de T_1, entonces la secuencia de ejecución es la de la Figura 15.4. De nuevo, como se esperaba, se conserva la suma $A + B$ y los valores finales de las cuentas A y B son de 850 € y de 2.150 € respectivamente.

T_1	T_2
leer(A)	
$A := A - 50$	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)

FIGURA 15.3. Planificación 1 —una planificación secuencial en la que T_2 sigue a T_1.

T_1	T_2
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)
leer(A)	
$A := A - 50$	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	

FIGURA 15.4. Planificación 2 —una planificación secuencial en la cual T_1 sigue a T_2.

Las secuencias de ejecución que se acaban de describir se denominan **planificaciones**. Representan el orden cronológico en el que se ejecutan las instrucciones en el sistema. Obviamente una planificación para un conjunto de transacciones debe consistir en todas las instrucciones de dichas transacciones, y debe conservar el orden en que aparecen las instrucciones en cada transacción individual. Por ejemplo, en la transacción T_1 la instrucción escribir(A) debe aparecer antes de la instrucción leer(B) en cualquier planificación válida. En los párrafos siguientes, planificación 1 se referirá a la primera secuencia de ejecución (T_1 seguida de T_2) y planificación 2 a la segunda secuencia de ejecución (T_2 seguida de T_1).

Estas planificaciones son **secuenciales**. Cada planificación secuencial consiste en una secuencia de instrucciones de varias transacciones, en la cual las instrucciones pertenecientes a una única transacción están juntas en dicha planificación. De este modo, para un conjunto de n transacciones existen $n!$ planificaciones secuenciales válidas distintas.

Cuando el sistema de bases de datos ejecuta concurrentemente varias transacciones, la planificación correspondiente no tiene por qué ser secuencial. Si dos transacciones se ejecutan concurrentemente, el sistema operativo puede ejecutar una transacción durante un tiempo pequeño, luego realizar un cambio de contexto, ejecutar la segunda transacción durante un tiempo, cambiar de nuevo a la primera transacción durante un tiempo y así sucesivamente. Si hay muchas transacciones, todas ellas comparten el tiempo de la UCP.

Son posibles muchas secuencias de ejecución, puesto que varias instrucciones de ambas transacciones se pueden intercalar. En general no es posible predecir exactamente cuántas instrucciones se ejecutarán antes de que la UCP cambie a otra transacción. Así, el número de planificaciones posibles para un conjunto de n transacciones es mucho mayor que $n!$

Volviendo al ejemplo anterior supóngase que las dos transacciones se ejecutan concurrentemente. Una posible planificación se muestra en la Figura 15.5. Una vez que la ejecución tiene lugar, se llega al mismo estado que cuando las transacciones se ejecutan secuencialmente en el orden T_1 seguida de T_2. La suma $A + B$ se conserva igualmente.

No todas las ejecuciones concurrentes producen un estado correcto. Como ejemplo de esto considérese la planificación de la Figura 15.6. Después de ejecutarse esta planificación se llega a un estado cuyos valores finales de las cuentas A y B son de 950 € y de 2.100 € respectivamente. Este estado final es un *estado inconsistente*, ya que se han ganado 50 € al procesar la ejecución concurrente. Realmente la ejecución de las dos transacciones no conserva la suma $A + B$.

T_1	T_2
leer(A) $A := A - 50$ escribir(A)	
	leer(A) $temp := A * 0.1$ $A := A - temp$ escribir(A)
leer(B) $B := B + 50$ escribir(B)	
	leer(B) $B := B + temp$ escribir(B)

FIGURA 15.5. Planificación 3 —una planificación concurrente equivalente a la planificación 1.

T_1	T_2
leer(A) $A := A - 50$	
	leer(A) $temp := A * 0.1$ $A := A - temp$ escribir(A) leer(B)
escribir(A) leer(B) $B := B + 50$ escribir(B)	
	$B := B + temp$ escribir(B)

FIGURA 15.6. Planificación 4 —una planificación concurrente.

Si se deja el control de la ejecución concurrente completamente al sistema operativo, son posibles muchas planificaciones, incluyendo las que dejan a la base de datos en un estado inconsistente, como la que se acaba de describir. Es una tarea del sistema de base de datos asegurar que cualquier planificación que se ejecute lleva a la base de datos a un estado consistente. El componente del sistema de base de datos que realiza esta tarea se denomina **componente de control de concurrencia**.

Se puede asegurar la consistencia de la base de datos en una ejecución concurrente si se está seguro de que cualquier planificación que se ejecute tiene el mismo efecto que otra que se hubiese ejecutado sin concurrencia. Es decir, la planificación debe ser, en cierto modo, equivalente a una planificación secuencial. Esta idea se examina en el Apartado 15.5.

15.5. SECUENCIALIDAD

El sistema de base de datos debe controlar la ejecución concurrente de las transacciones para asegurar que el estado de la base sigue siendo consistente. Antes de examinar cómo debe realizar esta tarea, el sistema de base de datos hay que entender primero las planificaciones que aseguran la consistencia y las que no.

Puesto que las transacciones son programas, es difícil calcular cuáles son las operaciones exactas que realiza una transacción y cómo interaccionan las operaciones de varias transacciones. Por este motivo no se van a interpretar los tipos de operaciones que puede realizar una transacción sobre un elemento de datos. En lugar de esto se consideran sólo dos operaciones: leer y escribir. Se asume así que entre una instrucción leer(Q) y otra escribir(Q) sobre un elemento de datos Q, una transacción puede realizar una secuencia arbitraria de operaciones con la copia Q que reside en la memoria intermedia local de dicha transacción. De este modo las únicas operaciones significativas de la transacción son, desde el punto de vista de la planificación, las instrucciones leer y escribir. Por tanto, normalmente sólo se mostrarán las instrucciones leer y escribir en las planificaciones, como se muestra en la planificación 3 de la Figura 15.7.

En este apartado se discuten diferentes formas de equivalencia de planificación; esto lleva a los conceptos de **secuencialidad en cuanto a conflictos** y **secuencialidad en cuanto a vistas**.

15.5.1. Secuencialidad en cuanto a conflictos

Considérese una planificación P en la cual hay dos instrucciones consecutivas I_i e I_j, pertenecientes a las transacciones T_i y T_j respectivamente ($i \neq j$). Si I_i e I_j se refieren a distintos elementos de datos se pueden intercambiar I_i e I_j sin afectar al resultado de cualquier instrucción de la planificación. Sin embargo, si I_i e I_j se

T_1	T_2
leer(A)	
escribir(A)	
	leer(A)
	escribir(A)
leer(B)	
escribir(B)	
	leer(B)
	escribir(B)

FIGURA 15.7. Planificación 3 —sólo se muestran las instrucciones leer y escribir.

refieren al mismo elemento Q entonces el orden de los dos pasos puede ser importante. Puesto que sólo se tienen en cuenta las instrucciones leer y escribir, se deben considerar cuatro casos:

1. I_i = leer(Q), I_j = leer(Q). El orden de I_i e I_j no importa, puesto que T_i y T_j leen el mismo valor de Q, independientemente del orden.
2. I_i = leer(Q), I_j = escribir(Q). Si I_i está antes que I_j, entonces T_i no lee el valor de Q que escribe la instrucción I_j de T_j. Si I_j está antes que I_i, entonces T_i lee el valor de Q escrito por T_j. Por tanto, el orden de I_i e I_j es importante.
3. I_i = escribir(Q), I_j = leer(Q). El orden de I_i e I_j es importante por razones similares a las del caso anterior.
4. I_i = escribir(Q), I_j = escribir(Q). Puesto que ambas instrucciones son operaciones escribir, el orden de dichas instrucciones no afecta ni a T_i ni a T_j. Sin embargo, el valor que obtendrá la siguiente instrucción leer(Q) de P sí se ve afectado, ya que únicamente se conserva en la base de datos la última de las dos instrucciones escribir. Si no hay ninguna otra instrucción escribir(Q) después de I_i e I_j en P, entonces el orden de I_i e I_j afecta directamente al valor final de Q en el estado de la base de datos que se obtiene con la planificación P.

De esta manera sólo en el caso en el cual I_i e I_j son instrucciones leer no tiene importancia el orden de ejecución de las mismas.

Se dice que I_i e I_j están en **conflicto** si existen operaciones de diferentes transacciones sobre el mismo elemento de datos, y al menos una de esas instrucciones es una operación escribir.

Para ilustrar el concepto de instrucciones conflictivas considérese la planificación 3 mostrada en la Figura 15.7. La instrucción escribir(A) de T_1 está en conflicto con la instrucción leer(A) de T_2. Sin embargo, la instrucción escribir(A) de T_2 no está en conflicto con la instrucción leer(B) de T_1, ya que las dos instrucciones acceden a diferentes elementos de datos.

Sean I_i e I_j instrucciones consecutivas de una planificación P. Si I_i e I_j son instrucciones de transacciones diferentes y además I_i e I_j no están en conflicto, entonces se puede cambiar el orden de I_i e I_j para obtener una nueva planificación P'. Lo esperado es que P sea equivalente a P', ya que todas las instrucciones aparecen en el mismo orden en ambas planificaciones salvo I_i e I_j, cuyo orden no es importante.

Puesto que la instrucción escribir(A) de T_2 en la planificación 3 de la Figura 15.7 no está en conflicto con la instrucción leer(B) de T_1, se pueden intercambiar dichas instrucciones para generar una planificación equivalente, la planificación 5, que se muestra en la Figura 15.8. Independientemente de cuál sea el estado inicial del sistema, las planificaciones 3 y 5 producen el mismo estado final del sistema.

Se puede continuar intercambiando instrucciones no conflictivas como sigue:

- Intercambiar la instrucción leer(B) de T_1 con la instrucción leer(A) de T_2.
- Intercambiar la instrucción escribir(B) de T_1 con la instrucción escribir(A) de T_2.
- Intercambiar la instrucción escribir(B) de T_1 con la instrucción leer(A) de T_2.

El resultado final de estos intercambios, como se muestra en la Figura 15.9, es una planificación secuencial. Así se muestra que la planificación 3 es equivalente a una planificación secuencial. Esta equivalencia implica que independientemente del estado inicial, la planificación 3 produce el mismo estado final que una planificación secuencial.

Si una planificación P se puede transformar en otra P' por medio de una serie de intercambios de instrucciones no conflictivas, se dice que P y P' son **equivalentes en cuanto a conflictos**.

Volviendo a los ejemplos anteriores, se observa que la planificación 1 no es equivalente en cuanto a conflictos a la planificación 2. Sin embargo, la planificación 1 es equivalente en cuanto a conflictos a la planificación 3, puesto que las instrucciones leer(B) y escribir(B) de T_1 se pueden intercambiar con las instrucciones leer(A) y escribir(A) de T_2.

El concepto de equivalencia en cuanto a conflictos lleva al concepto de secuencialidad en cuanto a con-

T_1	T_2
leer(A)	
escribir(A)	
	leer(A)
leer(B)	
	escribir(A)
escribir(B)	
	leer(B)
	escribir(B)

FIGURA 15.8. Planificación 5 —planificación 3 después de intercambiar un par de instrucciones.

T_1	T_2
leer(A)	
escribir(A)	
leer(B)	
escribir(B)	
	leer(A)
	escribir(A)
	leer(B)
	escribir(B)

FIGURA 15.9. Planificación 6 —una planificación secuencial equivalente a la planificación 3.

flictos. Se dice que una planificación P es **secuenciable en cuanto a conflictos** si es equivalente en cuanto a conflictos a una planificación secuencial. Así, la planificación 3 es secuenciable en cuanto a conflictos, ya que es equivalente en cuanto a conflictos a la planificación secuencial 1.

Finalmente considérese la planificación 7 de la Figura 15.10; sólo consta de las operaciones significativas de las transacciones T_3 y T_4 (es decir, leer y escribir). Esta planificación no es secuenciable en cuanto a conflictos, ya que no es equivalente ni a la planificación secuencial $<T_3,T_4>$ ni a $<T_4,T_3>$.

Es posible encontrar dos planificaciones que produzcan el mismo resultado y que no sean equivalentes en cuanto a conflictos. Por ejemplo, considérese la transacción T_5, la cual transfiere 10 € de la cuenta B a la A. Sea la planificación 8 la que se define en la Figura 15.11. Se puede afirmar que la planificación 8 no es equivalente en cuanto a conflictos a la planificación secuencial $<T_1,T_5>$, ya que en la planificación 8 la instrucción escribir(B) de T_5 está en conflicto con la instrucción leer(B) de T_1. Por este motivo no se pueden colocar todas las instrucciones de T_1 antes de las de T_5 intercambiando instrucciones no conflictivas consecutivas. Sin embargo, los valores finales de las cuentas A y B son los mismos después de ejecutar tanto la planificación 8 como la planificación secuencial $<T_1, T_5>$ —esto es, 960 € y 2.040 € respectivamente.

Con este ejemplo se puede observar que existen definiciones de equivalencia de planificaciones que son menos rigurosas que la de equivalencia en cuanto a conflictos. Para que el sistema pueda determinar que el resultado de la planificación sea el mismo que el de la planificación secuencial $<T_1, T_5>$, debe analizar los cálculos realizados por T_1 y T_5, en lugar de tener sólo en cuenta las operaciones leer y escribir. En general es difícil de implementar tal análisis y es caro en términos de cómputo. Sin embargo, existen otras definiciones de

T_3	T_4
leer(Q)	
	escribir(Q)
escribir(Q)	

FIGURA 15.10. Planificación 7.

T_1	T_5
leer(A)	
$A := A - 50$	
escribir(A)	
	leer(B)
	$B := B - 10$
	escribir(B)
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(A)
	$A := A + 10$
	escribir(A)

FIGURA 15.11. Planificación 8.

equivalencia de planificación que se basan únicamente en las operaciones leer y escribir. En el siguiente apartado se considera una de estas definiciones.

15.5.2. Secuencialidad en cuanto a vistas

En este apartado se va a considerar una forma de equivalencia que es menos rigurosa que la equivalencia en cuanto a conflictos, pero que, al igual que ésta, se basa únicamente en las operaciones leer y escribir de las transacciones.

Considérense dos planificaciones, P y P', en las cuales participa el mismo conjunto de transacciones. Se dice que las planificaciones P y P' son *equivalentes en cuanto a vistas* si se cumplen las tres condiciones siguientes:

1. Para todo elemento de datos Q, si la transacción T_i lee el valor inicial de Q en la planificación P, entonces T_i debe leer también el valor inicial de Q en la planificación P'.
2. Para todo elemento de datos Q, si la transacción T_i ejecuta leer(Q) en la planificación P y el valor lo ha producido la transacción T_j (si existe dicha transacción) entonces, en la planificación P', la transacción T_i debe leer también el valor de Q que haya producido la transacción T_j.
3. Para todo elemento de datos Q, la transacción (si existe) que realice la última operación escribir(Q) en la planificación P, debe realizar la última operación escribir(Q) en la planificación P'.

Las condiciones 1 y 2 aseguran que cada transacción lee los mismos valores en ambas planificaciones y por tanto realizan los mismo cálculos. La condición 3, junto con las condiciones 1 y 2, asegura que ambas planificaciones dan como resultado el mismo estado final del sistema.

Volviendo a los ejemplos anteriores, nótese que la planificación 1 no es equivalente en cuanto a vistas a la planificación 2, ya que en la planificación 1 el valor de

la cuenta A que lee la transacción T_2 lo produce T_1, mientras que esto no ocurre en la planificación 2. Sin embargo, la planificación 1 es equivalente en cuanto a vistas a la planificación 3, ya que los valores de las cuentas A y B que lee la transacción T_2 los produce T_1 en ambas planificaciones.

El concepto de equivalencia en cuanto a vistas lleva al concepto de secuencialidad en cuanto a vistas. Se dice que la planificación P es **secuenciable en cuanto a vistas** si es equivalente en cuanto a vistas a una planificación secuencial.

Como ejemplo supóngase que se aumenta la planificación 7 con la transacción T_6 y se obtiene la planificación 9, como se muestra en la Figura 15.12. La planificación 9 es secuenciable en cuanto a vistas. Realmente es equivalente en cuanto a vistas a la planificación secuencial $<T_3,T_4,T_6>$, ya que la instrucción leer(Q) lee el valor inicial de Q en ambas planificaciones, y T_6 realiza la escritura final de Q en ambas planificaciones.

Toda planificación secuenciable en cuanto a conflictos es secuenciable en cuanto a vistas, pero existen planificaciones secuenciables en cuanto a vistas que no son secuenciables en cuanto a conflictos. Realmente, la planificación 9 no es secuenciable en cuanto a conflictos, puesto que para todo par de instrucciones éstas están en conflicto y, por tanto, no es posible ningún intercambio de instrucciones.

Obsérvese que en la planificación 9, las transacciones T_4 y T_6 realizan operaciones escribir(Q) sin haber realizado ninguna operación leer(Q). Este tipo de escrituras se denominan **escrituras a ciegas**. Las escrituras a ciegas aparecen en toda planificación secuenciable en cuanto a vistas que no sea secuenciable en cuanto a conflictos.

15.6. RECUPERABILIDAD

Antes se han estudiado las planificaciones que son aceptables desde el punto de vista de la consistencia de la base de datos asumiendo implícitamente que no había fallos en las transacciones. Ahora se va a estudiar el efecto de los fallos en una transacción durante una ejecución concurrente.

Si la transacción T_i falla, por la razón que sea, es necesario deshacer el efecto de dicha transacción para asegurar la propiedad de atomicidad de la misma. En un sistema que permita la concurrencia es necesario asegurar también que toda transacción T_j que dependa de T_i (es decir, T_j lee datos que ha escrito T_i) se aborta también. Para alcanzar esta garantía, es necesario poner restricciones al tipo de planificaciones permitidas en el sistema.

En los dos subapartados siguientes se estudian las planificaciones que son aceptables desde el punto de vista que se ha descrito. Como ya se dijo antes, en el Capítulo 16 se describe la manera de asegurar que sólo se generan dichas planificaciones aceptables.

15.6.1. Planificaciones recuperables

Considérese la planificación 11 que se muestra en la Figura 15.13, en la cual la transacción T_9 realiza sólo una instrucción: leer(A). Supóngase que el sistema permite que T_9 se complete inmediatamente después de ejecutar la instrucción leer(A). Así se completa T_9 antes de que lo haga T_8. Supóngase ahora que T_8 falla antes de completarse. Puesto que T_9 ha leído el valor del elemento de datos A escrito por T_8, se debe abortar T_9 para asegurar la atomicidad de la transacción. Sin embargo, T_9 ya se ha comprometido y no puede abortarse. De este modo se llega a una situación en la cual es imposible recuperarse correctamente del fallo de T_8.

La planificación 11, cuyo compromiso tiene lugar inmediatamente después de ejecutar la instrucción leer(A), es un ejemplo de planificación *no recuperable*, la cual no debe permitirse. La mayoría de los sistemas de bases de datos requieren que todas las planificaciones sean *recuperables*. Una **planificación recuperable** es aquella en la que para todo par de transacciones T_i y T_j tales que T_j lee elementos de datos que ha escrito previamente T_i, la operación comprometer de T_i aparece antes que la de T_j.

15.6.2. Planificaciones sin cascada

Incluso si una planificación es recuperable, hay que retroceder varias transacciones para recuperar correctamente el estado previo a un fallo en una transacción T_i. Tales

T_3	T_4	T_6
leer(Q)		
	escribir(Q)	
escribir(Q)		
		escribir(Q)

FIGURA 15.12. Planificación 9 —una planificación secuenciable en cuanto a vistas.

T_8	T_9
leer(A)	
escribir(A)	
	leer(A)
leer(B)	

FIGURA 15.13. Planificación 11.

situaciones ocurren si las transacciones leen datos que ha escrito T_i. Como ejemplo considérese la planificación parcial de la Figura 15.14. La transacción T_{10} escribe un valor de A que lee la transacción T_{11}. La transacción T_{11} escribe un valor de A que lee la transacción T_{12}. Supóngase que en ese momento falla T_{10}. Se debe retroceder T_{10}. Puesto que T_{11} depende de T_{10}, se debe retroceder T_{11}. Puesto que T_{12} depende de T_{11}, se debe retroceder T_{12}. Este fenómeno en el cual un fallo en una única transacción provoca una serie de retrocesos de la transacción se denomina **retroceso en cascada**.

No es deseable el retroceso en cascada, ya que provoca un aumento significativo del trabajo necesario para deshacer cálculos. Es deseable restringir las planificaciones a aquellas en las que no puedan ocurrir retrocesos en cascada. Tales planificaciones se denominan planificaciones sin cascada. Una **planificación sin cascada** es aquella para la que todo par de transacciones T_i y T_j tales que T_j lee un elemento de datos que ha escrito previamente T_i, la operación comprometer de T_i aparece antes que la operación de lectura de T_j. Es sencillo comprobar que toda planificación sin cascada es también recuperable.

15.7. IMPLEMENTACIÓN DEL AISLAMIENTO

Antes se han visto las propiedades que debe tener una planificación para dejar a la base de datos en un estado consistente y para permitir fallos en la transacción que se puedan manejar de una manera segura. En concreto, las planificaciones que son secuenciables en cuanto a conflictos o secuenciables en cuanto a vistas y sin cascada cumplen estos requisitos.

Existen varios **esquemas de control de concurrencia** que se pueden utilizar para asegurar que, incluso si se ejecutan concurrentemente muchas transacciones, sólo se generen planificaciones aceptables sin tener en cuenta la forma en que el sistema operativo comparte en el tiempo los recursos (tales como el tiempo de UCP) entre las transacciones.

Como ejemplo trivial de esquema de control de concurrencia considérese éste: una transacción realiza un **bloqueo** en la base de datos completa antes de comenzar y lo libera después de haberse comprometido. Mientras una transacción mantiene el bloqueo, no se permite que ninguna otra lo obtenga, y todas ellas deben esperar hasta que se libere el bloqueo. Como resultado de esta política de bloqueo, sólo se puede ejecutar una transacción cada vez. Por tanto, sólo se generan planificaciones secuenciales. Éstas son trivialmente secuenciables y es sencillo probar que también son sin cascada.

Un esquema de control de concurrencia como éste produce un rendimiento pobre, ya que fuerza a que las transacciones esperen a que las transacciones precedentes terminen para poder comenzar. En otras palabras, produce un grado de concurrencia pobre. Como se ha explicado en el Apartado 15.4, la ejecución concurrente tiene varios beneficios en el rendimiento.

El objetivo de los esquemas de control de concurrencia es proporcionar un elevado grado de concurrencia, al mismo tiempo que aseguran que todas las planificaciones que se generan son secuenciables en cuanto a conflictos o en cuanto a vistas y son sin cascada.

En el Capítulo 16 se estudian gran número de esquemas de control de concurrencia. Los esquemas adoptan diferentes compromisos en función del aumento de concurrencia que permiten y del aumento de coste en el que incurren. Algunos permiten que se generen planificaciones secuenciables en cuanto a conflictos; otros permiten que se generen planificaciones secuenciables en cuanto a vistas que no son secuenciables en cuanto a conflictos.

15.8. DEFINICIÓN DE TRANSACCIONES EN SQL

Un lenguaje de manipulación de datos debe incluir una constructora para especificar el conjunto de acciones que constituyen una transacción.

T_{10}	T_{11}	T_{12}
leer(A)		
leer(B)		
escribir(A)		
	leer(A)	
	escribir(A)	
		leer(A)

FIGURA 15.14. Planificación 12.

En la norma SQL se especifica el comienzo de una transacción explícitamente. Las transacciones se terminan con una de las instrucciones SQL siguientes:

- **Commit work** compromete la transacción actual y comienza una nueva.
- **Rollback work** provoca que la transacción actual aborte.

La palabra clave **work** es opcional en ambas instrucciones. Si el programa termina sin ninguna de estas órdenes, los cambios o bien se comprometen o bien se retroceden; en la norma no se especifica cuál de las

dos acciones tiene lugar, y depende de la implementación.

La norma especifica también que el sistema debe garantizar tanto la secuencialidad como la ausencia de retroceso en cascada. La definición de secuencialidad que usa la norma es que la planificación debe tener el *mismo efecto* que tendría una planificación secuencial. De este modo son aceptables tanto la secuencialidad en cuanto a conflictos como la secuencialidad en cuanto a vistas.

La norma SQL-92 permite también que una transacción especifique que puede ejecutarse de forma que se convierta en no secuenciable con respecto a otras transacciones. En el Apartado 16.8 se estudian estos niveles más débiles de consistencia.

15.9. COMPROBACIÓN DE LA SECUENCIALIDAD

Cuando se diseñan esquemas de control de concurrencia hay que demostrar que las planificaciones que genera el esquema son secuenciables. Para hacer esto se debe entender primero la forma de determinar si, dada una planificación concreta P, es secuenciable.

Ahora se presenta un método simple y eficiente de determinar la secuencialidad en cuanto a conflictos de una planificación. Considérese una planificación P. Se construye un grafo dirigido, llamado **grafo de precedencia** para P. Este grafo consiste en un par $G = (V,A)$, siendo V un conjunto de vértices y A un conjunto de arcos. El conjunto de vértices consiste en todas las transacciones que participan en la planificación. El conjunto de arcos consiste en todos los arcos $T_i \rightarrow T_j$ para los cuales se dan una de las tres condiciones siguientes

1. T_i ejecuta escribir(Q) antes de que T_j ejecute leer(Q).
2. T_i ejecuta leer(Q) antes de que T_j ejecute escribir(Q).
3. T_i ejecuta escribir(Q) antes de que T_j ejecute escribir(Q).

Si existe un arco $T_i \rightarrow T_j$ en el grafo de precedencia, entonces en toda planificación secuencial P' equivalente a P, T_i debe aparecer antes de T_j.

Por ejemplo, en la Figura 15.15a se muestra el grafo de precedencia de la planificación 1. Sólo contiene el arco $T_1 \rightarrow T_2$, puesto que todas las instrucciones de T_1 se ejecutan antes de que lo haga la primera de T_2. Análogamente, la Figura 15.15b muestra el grafo de precedencia de la planificación 2 con el único arco $T_2 \rightarrow T_1$, ya que todas las instrucciones de T_2 se ejecutan antes de que lo haga la primera de T_1.

El grafo de precedencia de la planificación 4 se representa en la Figura 15.16. Contiene el arco $T_1 \rightarrow T_2$ debido a que T_1 ejecuta leer(A) antes de que T_2 ejecute escribir(A). También contiene el arco $T_2 \rightarrow T_1$ debido a que T_2 ejecuta leer(B) antes de que T_1 ejecute escribir(B).



FIGURA 15.15. Grafo de precedencia para (a) la planificación 1 y (b) la planificación 2.

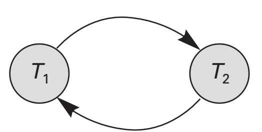


FIGURA 15.16. Grafo de precedencia para la planificación 4.

Si el grafo de precedencia de P tiene un ciclo, entonces la planificación P no es secuenciable en cuanto a conflictos. Si el grafo no contiene ciclos, entonces la planificación P es secuenciable en cuanto a conflictos.

El **orden de secuencialidad** se puede obtener a través de la **ordenación topológica**, la cual determina un orden lineal que consiste en el orden parcial del grafo de precedencia. En general se pueden obtener muchos órdenes lineales posibles a través de la ordenación topo-

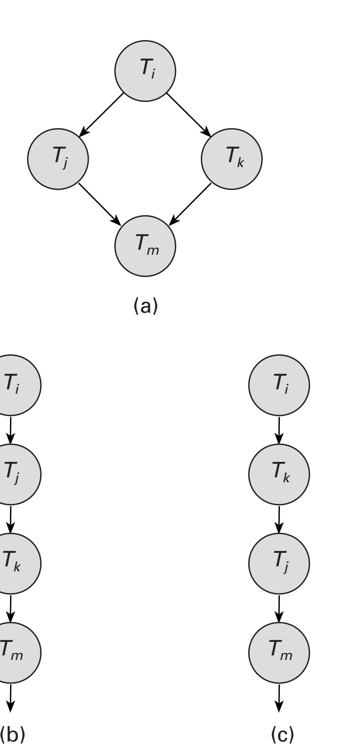


FIGURA 15.17. Ilustración de la ordenación topológica.

lógica. Por ejemplo, el grafo de la Figura 15.17a tiene dos órdenes lineales aceptables, como se observa en las figuras 15.17b y 15.17c.

Así, para probar la secuencialidad en cuanto a conflictos hay que construir el grafo de precedencia e invocar a un algoritmo de detección de ciclos. Los algoritmos de detección de ciclos se pueden encontrar en cualquier libro de texto sobre algoritmos. Los algoritmos de detección de ciclos, tales como los que se basan en la búsqueda primero en profundidad, requieren del orden de n^2 operaciones, siendo n el número de vértices del grafo (es decir, el número de transacciones). De este modo se tiene un esquema práctico para determinar la secuencialidad en cuanto a conflictos.

Volviendo a los ejemplos anteriores, nótese que los grafos de precedencia para las planificaciones 1 y 2 (Figura 15.15) no contienen ciclos. El grafo de precedencia para la planificación 4 (Figura 15.16), sin embargo, contiene ciclos, lo que indica que esta planificación no es secuenciable en cuanto a conflictos.

La comprobación de la secuencialidad en cuanto a vistas es más complicada. De hecho, se ha demostrado que el problema de determinar la secuencialidad en cuanto a vistas es *NP*-completo. Por tanto, seguramente no exista ningún algoritmo eficiente para comprobar la secuencialidad en cuanto a vistas. Véanse la notas bibliográficas para consultar referencias sobre ello. Sin embargo, los esquemas de control de concurrencia aún pueden utilizar *condiciones suficientes* para la secuencialidad en cuanto a vistas. Es decir, si se cumplen las condiciones suficientes, la planificación es secuencialidad en cuanto a vistas, pero puede haber algunas planificaciones secuencialidad en cuanto a vistas que no satisfagan las condiciones suficientes.

15.10. RESUMEN

- Una *transacción* es una *unidad* de la ejecución de un programa que accede y posiblemente actualiza varios elementos de datos. Es fundamental comprender el concepto de transacción para entender e implementar las actualizaciones de los datos en una base de datos, de manera que las ejecuciones concurrentes y los fallos de varios tipos no resulten en que la base de datos se vuelva inconsistente.
- Es necesario que las transacciones tengan las propiedades ACID: atomicidad, consistencia, aislamiento y durabilidad.
  — La atomicidad asegura que, o bien todos los efectos de la transacción se reflejan en la base de datos, o bien ninguno de ellos; un fallo no puede dejar a la base de datos en un estado en el cual una transacción se haya ejecutado parcialmente.
  — La consistencia asegura que si la base de datos es consistente inicialmente, la ejecución de la transacción (debido a la misma) deja la base de datos en un estado consistente.
  — El aislamiento asegura que en la ejecución concurrente de transacciones, están aisladas entre sí, de tal manera que cada una tiene la impresión de que ninguna otra transacción se ejecuta concurrentemente con ella.
  — La durabilidad asegura que, una vez que la transacción se ha comprometido, las actualizaciones hechas por la transacción no se pierden incluso si hay un fallo del sistema.
- La ejecución concurrente de transacciones mejora la productividad y la utilización del sistema, y también reduce el tiempo de espera de las transacciones.
- Cuando varias transacciones se ejecutan concurrentemente en la base de datos, puede que deje de conservarse la consistencia de los datos. Es, por tanto, necesario que el sistema controle la interacción entre las transacciones concurrentes.
  — Puesto que una transacción es una unidad que conserva la consistencia, una ejecución secuencial de transacciones garantiza que se conserva dicha consistencia.
  — Una *planificación* captura las acciones clave de las transacciones que afectan a la ejecución concurrente, tales como las operaciones leer y escribir, a la vez que se abstraen los detalles internos de la ejecución de la transacción.
  — Es necesario que toda planificación producida por el procesamiento concurrente de un conjunto de transacciones tenga el efecto equivalente a una planificación en la cual esas transacciones se ejecutan secuencialmente en un cierto orden.
  — Un sistema que asegure esta propiedad se dice que asegura la *secuencialidad*.
  — Existen varias nociones distintas de equivalencia que llevan a los conceptos de *secuencialidad en cuanto a conflictos* y *secuencialidad en cuanto a vistas*.
- Se puede asegurar la secuencialidad de las planificaciones generadas por la ejecución concurrente de transacciones por medio de una gran variedad de mecanismos llamados esquemas de *control de concurrencia*.
- Las planificaciones deben ser recuperables para asegurar que si la transacción *a* ve los efectos de la transacción *b* y ésta aborta, entonces *a* también se aborte.
- Las planificaciones deben ser preferentemente sin cascada para que el hecho de abortar una transacción no provoque en abortos en cascada de otras transacciones.

- El componente de gestión de control de concurrencia de la base de datos es el responsable de manejar los esquemas de control de concurrencia. En el Capítulo 16 se describen esquemas de control de concurrencia.
- El componente de gestión de recuperaciones de la base de datos es el responsable de asegurar las propiedades de las transacciones de atomicidad y durabilidad.

  El esquema de copia en la sombra se usa para asegurar la atomicidad y durabilidad en los editores de texto; sin embargo, tiene sobrecargas extremadamente altas cuando se usa en los sistemas de bases de datos y, más aún, no soporta la ejecución concurrente. El Capítulo 17 trata esquemas mejores.
- Se puede comprobar si una planificación es secuenciable en cuanto a conflictos construyendo el *grafo de precedencia* para dicha planificación y viendo que no hay ciclos en el grafo. Sin embargo, hay esquemas de control de concurrencia más eficientes para asegurar la secuencialidad.

TÉRMINOS DE REPASO

- Bloqueo
- Conflictos entre operaciones
- Determinación de la secuencialidad
- Ejecución secuencial
- Ejecuciones concurrentes
- Equivalencia en cuanto a conflictos
- Equivalencia en cuanto a vistas
- Escrituras a ciegas
- Escrituras externas observables
- Esquema de control de concurrencia
- Esquema de copia en la sombra
- Estado inconsistente
- Estados de una transacción
  - Abortada
  - Activa
  - Comprometida
  - Fallida
  - Parcialmente comprometida
  - Terminada
- Grafo de precedencia
- Orden de secuencialidad
- Planificaciones
- Planificaciones recuperables
- Planificaciones sin cascada
- Propiedades ACID
  - Aislamiento
  - Atomicidad
  - Consistencia
  - Durabilidad
- Recuperabilidad
- Retroceso en cascada
- Secuencialidad en cuanto a conflictos
- Secuencialidad en cuanto a vistas
- Transacción
- Transacción
  - Cancelar
  - Reiniciar

EJERCICIOS

15.1. Lístense las propiedades ACID. Explíquese la utilidad de cada una.

15.2. Supóngase que existe un sistema de base de datos que nunca falla. ¿Se necesita un gestor de recuperaciones para este sistema?

15.3. Considérese un sistema de archivos como el de su sistema operativo preferido.

a. ¿Cuáles son los pasos involucrados en la creación y borrado de archivos, y en la escritura de datos a archivos?

b. Explíquese por qué son relevantes los aspectos de atomicidad y durabilidad en la creación y borrado de archivos, y en la escritura de datos a archivos.

Razónese la respuesta.

15.4. Los implementadores de sistemas de bases de datos prestan mucha más atención a las propiedades ACID que los implementadores de sistemas de archivos. ¿Por qué tiene sentido esto?

15.5. Durante su ejecución, una transacción pasa a través de varios estados hasta que se compromete o aborta. Lístense todas las secuencias posibles de estados por los que puede pasar una transacción. Explíquese por qué puede ocurrir cada una de las transiciones de estados.

15.6. Justifíquese lo siguiente. La ejecución concurrente de transacciones es más importante cuando los datos se deben extraer de disco (lento) o cuando las transacciones duran mucho, y es menos importante cuando hay pocos datos en memoria y las transacciones son muy cortas.

15.7. Explíquese la diferencia entre los términos *planificación secuencial* y *planificación secuenciable*.

15.8. Considérense las dos transacciones siguientes:

T_1: leer(A);
leer(B);
si $A = 0$ **entonces** $B := B + 1$;
escribir(B).

T_2: leer(B);
leer(A);
si $B = 0$ **entonces** $A := A + 1$;
escribir(A).

Sea el requisito de consistencia $A = 0 \vee B = 0$, siendo los valores iniciales $A = B = 0$.

a. Demuéstrese que toda ejecución secuencial en la que aparezcan estas transacciones conserva la consistencia de la base de datos.

b. Muéstrese una ejecución concurrente de T_1 y T_2 que produzca una planificación no secuenciable.

c. ¿Existe una ejecución concurrente de T_1 y T_2 que produzca una planificación secuenciable?

15.9. Puesto que toda planificación secuenciable en cuanto a conflictos es secuenciable en cuanto a vistas, ¿por qué se hace hincapié en la secuencialidad en cuanto a conflictos en vez de en la secuencialidad en cuanto a vistas?

15.10. Considérese el grafo de precedencia de la Figura 15.18. ¿Es secuenciable en cuanto a conflictos la planificación correspondiente? Razónese la respuesta.

15.11. ¿Qué es una planificación recuperable? ¿Por qué es conveniente la recuperabilidad de las planificaciones? ¿Hay circunstancias bajo las cuales puede ser conveniente permitir planificaciones no recuperables? Razónese la respuesta.

15.12. ¿Qué es una planificación sin cascada? ¿Por qué es conveniente la planificación sin cascada? ¿Hay circunstancias bajo las cuales puede ser conveniente permitir planificaciones que no sean sin cascada? Razónese la respuesta.

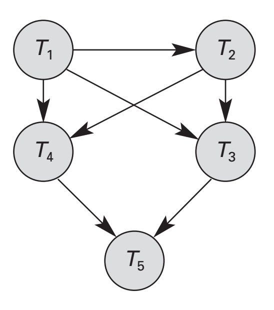


FIGURA 15.18. Grafo de precedencia.

NOTAS BIBLIOGRÁFICAS

Gray y Reuter [1993] proporcionan un extenso tratamiento de los conceptos y técnicas de procesamiento de transacciones, técnicas y detalles de implementación, incluyendo resultados de recuperación y control de concurrencia. Bernstein y Newcomer [1997] proporcionan varios aspectos del procesamiento de transacciones.

Las primeras discusiones en libro de texto del control de concurrencia y la recuperación se incluyen en Papadimitriou [1986] y Bernstein et al. [1987]. En Gray [1978] se presenta una visión de conjunto sobre resultados de implementación de control de concurrencia y recuperación.

El concepto de secuencialidad se formuló en Eswaran et al. [1976] en conexión con su trabajo sobre control de concurrencia para System R. Los resultados sobre la determinación de secuencialidad y NP-completitud de la determinación de la secuencialidad en cuanto a vistas son de Papadimitriou et al. [1977] y Papadimitriou [1979]. Los algoritmos de detección de ciclos se pueden encontrar en libros de algoritmos estándar, como en Comen et al. [1990].

En los Capítulos 16, 17 y 24 se citan referencias de aspectos específicos sobre el procesamiento de transacciones, como el control de concurrencia y la recuperación.

CAPÍTULO

16 CONTROL DE CONCURRENCIA

EN el Capítulo 15 se vio que una de las propiedades fundamentales de las transacciones es el aislamiento. Cuando se ejecutan varias transacciones concurrentemente en la base de datos, puede que deje de conservarse la propiedad de aislamiento. Es necesario que el sistema controle la interacción entre las transacciones concurrentes; dicho control se lleva a cabo a través de uno de los muchos mecanismos existentes llamado esquemas de *control de concurrencia*.

Todos los esquemas de control de concurrencia que se describen en este capítulo se basan en la propiedad de secuencialidad. Es decir, todos los esquemas que se presentan aquí aseguran que las planificaciones son secuenciables. En el Capítulo 24 se describen esquemas de control de concurrencia que admiten planificaciones no secuenciables. En este capítulo se trata la gestión de la ejecución concurrente de transacciones y se ignoran los fallos. En el Capítulo 17 se verá la manera en que el sistema se puede recuperar de los fallos.

16.1. PROTOCOLOS BASADOS EN EL BLOQUEO

Una forma de asegurar la secuencialidad es exigir que el acceso a los elementos de datos se haga en exclusión mutua; es decir, mientras una transacción accede a un elemento de datos, ninguna otra transacción puede modificar dicho elemento. El método más habitual que se usa para implementar este requisito es permitir que una transacción acceda a un elemento de datos sólo si posee actualmente un bloqueo sobre dicho elemento.

16.1.1. Bloqueos

Existen muchos modos mediante los cuales se puede bloquear un elemento de datos. En este apartado se centra la atención en dos de dichos modos:

1. **Compartido.** Si una transacción T_i obtiene un **bloqueo en modo compartido** (denotado por C) sobre el elemento Q, entonces T_i puede leer Q pero no lo puede escribir.
2. **Exclusivo.** Si una transacción T_i obtiene un **bloqueo en modo exclusivo** (denotado por X) sobre el elemento Q, entonces T_i puede tanto leer como escribir Q.

Es necesario que toda transacción **solicite** un bloqueo del modo apropiado sobre el elemento de datos Q dependiendo de los tipos de operaciones que se vayan a realizar sobre Q. La petición se hace al gestor de control de concurrencia. La transacción puede realizar la operación sólo después de que el gestor de control de concurrencia **conceda** el bloqueo a la transacción.

Dado un conjunto de modos de bloqueo, se puede definir sobre ellos una **función de compatibilidad** como sigue. Utilizamos A y B para representar dos modos de bloqueo arbitrarios. Supóngase que la transacción T_i solicita un bloqueo en modo A sobre el elemento Q, en el que la transacción T_j ($T_i \neq T_j$) posee actualmente un bloqueo de modo B. Si a la transacción T_i se le puede conceder un bloqueo sobre Q a pesar de la presencia del bloqueo de modo B, entonces se dice que el modo A es **compatible** con el modo B. Tal función se puede representar convenientemente en forma de matriz. La relación de compatibilidad entre los dos modos de bloqueo que se usan en este apartado se presenta en la matriz comp de la Figura 16.1. Un elemento comp(A, B) de la matriz tiene el valor *cierto* si y sólo si el modo A es compatible con el modo B.

Nótese que el modo compartido es compatible con otro modo compartido, pero no con el modo exclusivo. En todo momento se pueden tener varios bloqueos en modo compartido (por varias transacciones) sobre un elemento de datos en concreto. Una petición posterior de bloqueo en modo exclusivo debe esperar hasta que se liberen los bloqueos en modo compartido que se poseen actualmente.

	C	X
C	cierto	falso
X	falso	falso

FIGURA 16.1. Matriz de compatibilidad de bloqueo comp.

Una transacción solicita un bloqueo compartido sobre el elemento de datos Q a través de la instrucción bloquear-C(Q). De forma similar se solicita un bloqueo exclusivo a través de la instrucción bloquear-X(Q). Se puede desbloquear un elemento de datos Q por medio de la instrucción desbloquear(Q).

Para acceder a un elemento de datos, una transacción T_i debe en primer lugar bloquear dicho elemento. Si éste ya se encuentra bloqueado por otra transacción en un modo incompatible, el gestor de control de concurrencia no concederá el bloqueo hasta que todos los bloqueos incompatibles que posean otras transacciones hayan sido liberados. De este modo T_i debe **esperar** hasta que se liberen todos los bloqueos incompatibles que posean otras transacciones.

La transacción T_i puede desbloquear un elemento de datos que haya bloqueado en algún momento anterior. Nótese que la transacción debe poseer un bloqueo sobre un elemento de datos durante todo el tiempo que acceda a dicho elemento. Además, no siempre es aconsejable que una transacción desbloquee un elemento de datos inmediatamente después de finalizar su acceso sobre él, ya que puede dejar de asegurarse la secuencialidad.

Como ejemplo, considérese de nuevo el sistema bancario simplificado que se presentó en el Capítulo 15. Sean A y B dos cuentas a las que acceden las transacciones T_1 y T_2. La transacción T_1 transfiere 50 € desde la cuenta B a la A (Figura 16.2). La transacción T_2 visualiza la cantidad total de dinero de las cuentas A y B —es decir, la suma $A + B$ (Figura 16.3).

Supóngase que los valores de las cuentas A y B son 100 € y 200 € respectivamente. Si estas dos transacciones se ejecutan secuencialmente, tanto en el orden T_1, T_2 como en el orden T_2, T_1, entonces la transacción T_2 visualizará el valor 300 €. Si por el contrario estas transacciones se ejecutan concurrentemente, entonces puede darse la planificación 1, que se muestra en la Figura 16.4. En ese caso la transacción T_2 visualiza 250 €, lo cual es incorrecto. El motivo por el que se produce esta incorrección es que la transacción T_1 desbloquea el elemento B demasiado pronto, lo cual provoca que T_2 perciba un estado inconsistente.

La planificación muestra las acciones que ejecuta cada transacción, así como los puntos en los que el gestor de control de concurrencia concede los bloqueos. La transacción que realiza una petición de bloqueo no pue-

```
leer(B);
B := B – 50;
escribir(B);
desbloquear(B);
bloquear-X(A);
leer(A);
A := A + 50;
escribir(A);
desbloquear
```

FIGURA 16.2. Transacción T_1.

```
T2: bloquear-C(A);
    leer(A);
    desbloquear(A);
    bloquear-C(B);
    leer(B);
    desbloquear(B);
    visualizar(A + B).
```

FIGURA 16.3. Transacción T_2.

de ejecutar su siguiente acción hasta que el gestor de control de concurrencia conceda dicho bloqueo. Por tanto, el bloqueo debe concederse en el intervalo de tiempo entre la operación de petición del bloqueo y la siguiente acción de la transacción. No es importante el momento exacto del intervalo en el cual se produce la concesión del bloqueo; se puede asumir sin problemas que la concesión del bloqueo se produce justo antes de la siguiente acción de la transacción. Por tanto se va a obviar la columna que describe las acciones del gestor de control de concurrencia en todas las planificaciones que aparezcan en el resto del capítulo. Se deja al lector como ejercicio la deducción del momento en que se conceden los bloqueos.

Supóngase ahora que el desbloqueo se retrasa hasta el final de la transacción. La transacción T_3 corresponde a T_1 con el desbloqueo retrasado (Figura 16.5). La transacción T_4 corresponde a T_2 con el desbloqueo retrasado (Figura 16.6).

Se puede verificar que la secuencia de lecturas y escrituras de la planificación 1, que provoca que se visualice un total incorrecto de 250 €, ya no es posible con T_3 y T_4.

Son posibles otras planificaciones. T_4 no visualiza un resultado inconsistente en ninguna de ellas; más adelante se verá por qué.

Por desgracia, el uso de bloqueos puede conducir a una situación no deseada. Considérese la planificación parcial de la Figura 16.7 para T_3 y T_4. Puesto que T_3 posee en bloqueo sobre B en modo exclusivo y T_4 solicita un bloqueo sobre B en modo compartido, T_4 espera a que T_3 desbloquee B. De forma similar, puesto que T_4 posee un bloqueo sobre A en modo compartido y T_3 solicita un bloqueo sobre A en modo exclusivo, T_3 espera a que T_4 desbloquee A. Así se llega a un estado en el cual ninguna de las transacciones puede continuar su ejecución normal. Esta situación se denomina **interbloqueo**. Cuando aparece un interbloqueo, el sistema debe retroceder una de las dos transacciones. Una vez que una de ellas se ha retrocedido, se desbloquean los elementos de datos que estuvieran bloqueados por la transacción. Estos elementos de datos están disponibles entonces para otra transacción, la cual puede continuar su ejecución. Se volverá al tratamiento de interbloqueos en el Apartado 16.6.

Si no se usan bloqueos, o se desbloquean los elementos de datos tan pronto como sea posible después de leerlos o escribirlos, se pueden obtener estados

T_1	T_2	Gestor de control de concurrencia
bloquear-X(B)		
		conceder-X(B, T_1)
leer(B)		
$B := B - 50$		
escribir(B)		
desbloquear(B)		
	bloquear-C(A)	
		conceder-C(A, T_2)
	leer(A)	
	desbloquear(A)	
	bloquear-C(B)	
		conceder-C(B, T_2)
	leer(B)	
	desbloquear(B)	
	visualizar($A + B$)	
bloquear-X(A)		
		conceder-X(A, T_1)
leer(A)		
$A := A + 50$		
escribir(A)		
desbloquear(A)		

FIGURA 16.4. Planificación 1.

inconsistentes. Por otro lado, si no se desbloquea un elemento de datos antes de solicitar un bloqueo sobre otro, pueden producirse interbloqueos. Existen formas de evitar los interbloqueos en algunas situaciones, como se verá en el Apartado 16.1.5. Sin embargo, en general, los interbloqueos son un mal necesario asociado a los bloqueos si se quieren evitar los estados inconsistentes. Los interbloqueos son absolutamente preferibles a los estados inconsistentes, ya que se pueden tratar retrocediendo las transacciones, mientras que los estados inconsistentes producen problemas en el mundo real que el sistema de base de datos no puede manejar.

Se exige que toda transacción del sistema siga un conjunto de reglas llamado **protocolo de bloqueo**, que indica el momento en que una transacción puede bloquear y desbloquear cada uno de los elementos de datos. Los protocolos de bloqueo restringen el número de planificaciones posibles. El conjunto de tales planificaciones es un subconjunto propio de todas las planificaciones secuenciables posibles. Se van a mostrar varios protocolos de bloqueo que sólo permiten planificaciones secuenciables en cuanto a conflictos. Antes de hacer esto se necesitan algunas definiciones.

Sean $\{T_0, T_1, \ldots, T_n\}$ un conjunto de transacciones que participan en la planificación S. Se dice que T_i **precede a** T_j en S, denotado por $T_i \rightarrow T_j$, si existe un elemento de datos Q tal que T_i ha obtenido un bloqueo en modo A sobre Q, y T_j ha obtenido un bloqueo en modo B sobre Q más tarde y comp(A,B) = falso. Si $T_i \rightarrow T_j$ entonces esta precedencia implica que en cualquier planificación secuencial equivalente, T_i debe aparecer antes que T_j. Obsérvese que este grafo es similar al grafo de precedencia que se usaba en el Apartado 15.9 para comprobar la secuencialidad en cuanto a conflictos. Los conflictos entre instrucciones corresponden a modos de bloqueo no compatibles.

Se dice que una planificación S es **legal** bajo un protocolo de bloqueo dado si S es una planificación posible para un conjunto de transacciones que sigan las reglas del protocolo de bloqueo. Se dice que un protocolo **asegura** la secuencialidad en cuanto a conflictos si y sólo si todas las planificaciones legales son secuenciables en cuanto a conflictos; en otras palabras, para todas las planificaciones legales la relación → asociada es acíclica.

T_3: bloquear-X(B);
leer(B);
$B := B - 50$;
escribir(B);
bloquear-X(A);
leer(A);
$A := A + 50$;
escribir(A);
desbloquear(B);
desbloquear(A).

FIGURA 16.5. Transacción T_3.

T_4: bloquear-C(A);
leer(A);
bloquear-C(B);
leer(B);
visualizar($A + B$).
desbloquear(A);
desbloquear(B).

FIGURA 16.6. Transacción T_4.

T_3	T_4
bloquear-X(B) leer(B) $B := B - 50$ escribir(B)	
	bloquear-C(A) leer(A) bloquear-C(B)
bloquear-X(A)	

FIGURA 16.7. Planificación 2.

16.1.2. Concesión de bloqueos

Cuando una transacción solicita un bloqueo de un modo particular sobre un elemento de datos y ninguna otra transacción posee un bloqueo sobre el mismo elemento de datos en un modo conflictivo, se puede conceder el bloqueo. Sin embargo, hay que tener cuidado para evitar la siguiente situación. Supóngase que la transacción T_2 posee un bloqueo en modo compartido sobre un elemento de datos y que la transacción T_1 solicita un bloqueo en modo exclusivo sobre dicho elemento de datos. Obviamente T_1 debe esperar a que T_2 libere el bloqueo en modo compartido. Mientras tanto, la transacción T_3 puede solicitar un bloqueo en modo compartido sobre el mismo elemento de datos. La petición de bloqueo es compatible con el bloqueo que se ha concedido a T_2, por tanto se puede conceder a T_3 el bloqueo en modo compartido. En este punto T_2 puede liberar el bloqueo, pero T_1 debe seguir esperando hasta que T_3 termine. Pero de nuevo puede haber una nueva transacción T_4 que solicite un bloqueo en modo compartido sobre el mismo elemento de datos y a la cual se conceda el bloqueo antes de que T_3 lo libere. De hecho es posible que haya una secuencia de transacciones que soliciten un bloqueo en modo compartido sobre el elemento de datos, y que cada una de ellas libere el bloqueo un poco después de que sea concedido, de forma que T_1 nunca obtenga el bloqueo en modo exclusivo sobre el elemento de datos. La transacción T_1 nunca progresa y se dice que tiene **inanición**.

Se puede evitar la inanición de las transacciones al conceder los bloqueos de la siguiente manera: cuando una transacción T_i solicita un bloqueo sobre un elemento de datos Q en un modo particular M, el gestor de control de concurrencia concede el bloqueo siempre que

1. No exista otra transacción que posea un bloqueo sobre Q en un modo que esté en conflicto con M.
2. No exista otra transacción que esté esperando un bloqueo sobre Q y que lo haya solicitado antes que T_i.

De este modo, una petición de bloqueo nunca se quedará bloqueada por otra petición de bloqueo solicitada más tarde.

16.1.3. Protocolo de bloqueo de dos fases

Un protocolo que asegura la secuencialidad es el **protocolo de bloqueo de dos fases**. Este protocolo exige que cada transacción realice las peticiones de bloqueo y desbloqueo de dos fases:

1. **Fase de crecimiento.** Una transacción puede obtener bloqueos pero no puede liberarlos.
2. **Fase de decrecimiento.** Una transacción puede liberar bloqueos pero no puede obtener ninguno nuevo.

Inicialmente una transacción está en la fase de crecimiento. La transacción adquiere los bloqueos que necesite. Una vez que la transacción libera un bloqueo, entra en la fase de decrecimiento y no puede realizar más peticiones de bloqueo.

Por ejemplo, las transacciones T_3 y T_4 son de dos fases. Por otro lado, las transacciones T_1 y T_2 no son de dos fases. Nótese que no es necesario que las instrucciones de desbloqueo aparezcan al final de la transacción. Por ejemplo, en el caso de la transacción T_3 se puede trasladar la instrucción desbloquear(B) hasta justo antes de la instrucción bloquear-X(A) y se sigue cumpliendo la propiedad del bloqueo de dos fases.

Se puede mostrar que el protocolo de bloqueo de dos fases asegura la secuencialidad en cuanto a conflictos. Considérese cualquier transacción. El punto de la planificación en el cual la transacción obtiene su bloqueo final (el final de la fase de crecimiento) se denomina **punto de bloqueo** de la transacción. Ahora se pueden ordenar las transacciones en base a sus puntos de bloqueo; de hecho, esta ordenación es un orden de secuencialidad para las transacciones. La demostración se deja como ejercicio para el lector (véase el Ejercicio 16.1).

El protocolo de bloqueo de dos fases *no* asegura la ausencia de interbloqueos. Obsérvese que las transacciones T_3 y T_4 son de dos fases pero en la planificación 2 (Figura 16.7) llegan a un interbloqueo.

Recuérdese del Apartado 15.6.2 que las planificaciones, además de ser secuenciables, es aconsejable que sean sin cascada. El retroceso en cascada puede ocurrir en el bloqueo de dos fases. Como ejemplo considérese la planificación parcial de la Figura 16.8. Cada transacción sigue el protocolo de bloqueo de dos fases pero un fallo de T_5 después del paso leer(A) de T_7 lleva a un retroceso en cascada de T_6 y T_7.

Los retrocesos en cascada se pueden evitar por medio de una modificación del protocolo de bloqueo de dos fases que se denomina **protocolo de bloqueo estricto de dos fases**. Este protocolo exige que, además de que el bloqueo sea de dos fases, una transacción debe poseer todos los bloqueos en modo exclusivo que tome hasta que dicha transacción se complete. Este requisito asegura que todo dato que escribe una transacción no comprometida está bloqueado en modo exclusivo has-

T_5	T_6	T_7
bloquear-X(A) leer(A) bloquear-C(B) leer(B) escribir(A) desbloquear(A)		
	bloquear-X(A) leer(A) escribir(A) desbloquear(A)	
		bloquear-C(A) leer(A)

FIGURA 16.8. Planificación parcial con bloqueo de dos fases.

ta que la transacción se completa, evitando que ninguna otra transacción lea el dato.

Otra variante del bloqueo de dos fases es el **protocolo de bloqueo riguroso de dos fases**, el cual exige que se posean todos los bloqueos hasta que se comprometa la transacción. Se puede comprobar fácilmente que con el bloqueo riguroso de dos fases se pueden secuenciar las transacciones en el orden en que se comprometen. Muchos sistemas de bases de datos implementan tanto el bloqueo estricto como el bloqueo estricto de dos fases.

Considérense las dos transacciones siguientes de las que sólo se muestran algunas de las operaciones leer y escribir significativas:

T_8: leer(a_1);
leer(a_2);
…
leer(a_n);
escribir(a_1).

T_9: leer(a_1);
leer(a_2);
visualizar($a_1 + a_2$).

Si se emplea el protocolo de bloqueo de dos fases entonces T_8 debe bloquear a_1 en modo exclusivo. Por tanto, toda ejecución concurrente de ambas transacciones conduce a una ejecución secuencial. Nótese sin embargo que T_8 sólo necesita el bloqueo en modo exclusivo sobre a_1 al final de su ejecución, cuando escribe a_1. Así, si T_8 pudiera bloquear inicialmente a_1 en modo compartido y después pudiera cambiar el bloqueo a modo exclusivo, se obtendría una mayor concurrencia, ya que T_8 y T_9 podrían acceder a a_1 y a_2 simultáneamente.

Esta observación lleva a un refinamiento del protocolo de bloqueo de dos fases básico en el cual se permiten **conversiones de bloqueo**. Se va a proporcionar un mecanismo para cambiar un bloqueo compartido por un bloqueo exclusivo y un bloqueo exclusivo por uno compartido. Se denota la conversión del modo compartido al modo exclusivo como **subir**, y la conversión del modo exclusivo al modo compartido como **bajar**. No se puede permitir la conversión de modos arbitrariamente. Por el contrario, la subida puede tener lugar sólo en la fase de crecimiento, mientras que la bajada puede tener lugar sólo en la fase de decrecimiento.

Volviendo al ejemplo, las transacciones T_8 y T_9 se pueden ejecutar concurrentemente bajo el protocolo de bloqueo de dos fases refinado, como muestra la planificación incompleta de la Figura 16.9, en la cual sólo se muestran algunas de las operaciones de bloqueo.

Nótese que se puede forzar a esperar a una transacción que intente subir un bloqueo sobre un elemento Q. Esta espera forzada tiene lugar si Q está bloqueado actualmente por *otra* transacción en modo compartido.

Del mismo modo que el protocolo de bloqueo de dos fases básico, el bloqueo de dos fases con conversión de bloqueos genera sólo planificaciones secuenciables en cuanto a conflictos y se pueden secuenciar las transacciones según sus puntos de bloqueo. Además, si se poseen los bloqueos hasta el final de la transacción, las planificaciones son sin cascada.

Para un conjunto de transacciones puede haber planificaciones secuenciables en cuanto a conflictos que no se puedan obtener por medio del protocolo de bloqueo de dos fases. Sin embargo, para obtener planificaciones secuenciables en cuanto a conflictos por medio de protocolos de bloqueo que no sean de dos fases, es necesario o bien tener información adicional de las transacciones o bien imponer alguna estructura u orden en el conjunto de elementos de datos de la base de datos. En ausencia de esta información es necesario el bloqueo de dos fases para la secuencialidad en cuanto a conflictos —si T_i no es una transacción de dos fases, siempre es posible encontrar otra transacción T_j que sea de dos fases tal que existe una planificación posible para T_i y T_j que no sea secuenciable en cuanto a conflictos.

Los bloqueos estricto de dos fases y riguroso de dos fases (con conversión de bloqueos) se usan ampliamente en sistemas de bases de datos comerciales.

Un esquema simple pero de uso extendido genera automáticamente las instrucciones apropiadas de bloqueo y desbloqueo para una transacción, basándose en peticiones de lectura y escritura desde la transacción:

T_8	T_9
bloquear-C(a_1)	
	bloquear-C(a_1)
bloquear-C(a_2)	
	bloquear-C(a_2)
bloquear-C(a_3) bloquear-C(a_4)	
	desbloquear(a_1) desbloquear(a_2)
bloquear-C(a_n) subir(a_1)	

FIGURA 16.9. Planificación incompleta con conversión de bloqueos.

- Cuando una transacción T_i realiza una operación leer(Q), el sistema genera una instrucción bloquear-C(Q) seguida de una instrucción leer(Q).
- Cuando T_i realiza una operación escribir(Q), el sistema comprueba si T_i posee ya un bloqueo en modo compartido sobre Q. Si es así, entonces el sistema genera una instrucción subir(Q) seguida de la instrucción escribir(Q). En otro caso el sistema genera una instrucción bloquear-X(Q) seguida de la instrucción escribir(Q).
- Todos los bloqueos que obtenga una transacción no se desbloquean hasta que dicha transacción se comprometa o aborte.

16.1.4. Implementación de bloqueos **

Un **gestor de bloqueos** se puede implementar como un proceso que recibe mensajes de transacciones y envía mensajes como respuesta. El proceso gestor de bloqueos responde a los mensajes de solicitud de bloqueo con mensajes de concesión de bloqueo, o con mensajes que solicitan un retroceso de la transacción (en caso de interbloqueos). Los mensajes de desbloqueo tan sólo requieren un reconocimiento como respuesta, pero pueden dar lugar a un mensaje de concesión para otra transacción que esté esperando.

El gestor de bloqueos utiliza la siguiente estructura de datos: para cada elemento de datos que está actualmente bloqueado, mantiene una lista enlazada de registros, uno para cada solicitud, en el orden en el que llegaron las solicitudes. Utiliza una tabla de asociación, indexada por el nombre del elemento de datos, para encontrar la lista enlazada (si la hay) para cada elemento de datos; esta tabla se denomina **tabla de bloqueos**. Cada registro de la lista enlazada de cada elemento de datos anota qué transacción hizo la solicitud, y qué modo de bloqueo se solicitó. El registro también anota si la solicitud ya se ha concedido.

La Figura 16.10 muestra un ejemplo de una tabla de bloqueos. La tabla contiene bloqueos para cinco elementos de datos distintos: E4, E7, E23, E44 y E912. La tabla de bloqueos utiliza cadenas de desbordamiento, de forma que hay una lista enlazada de elementos de datos por cada entrada de la tabla de bloqueos. También hay una lista de transacciones a las que se les han concedido bloqueos, o que están esperando para bloquear, para cada uno de los elementos de datos. Los bloqueos concedidos se han representado como rectángulos rellenos (negros), mientras que las solicitudes en espera son los rectángulos vacíos. Se ha omitido el modo de bloqueo para que la figura sea más sencilla. Por ejemplo, se puede observar que a T23 se le han concedido bloqueos sobre E912 y E7, y está esperando para bloquear E4.

Aunque la figura no lo muestre, la tabla de bloqueos debería mantener también un índice de identificadores de transacciones, de forma que fuese posible determinar de manera eficiente el conjunto de bloqueos que mantiene una transacción dada.

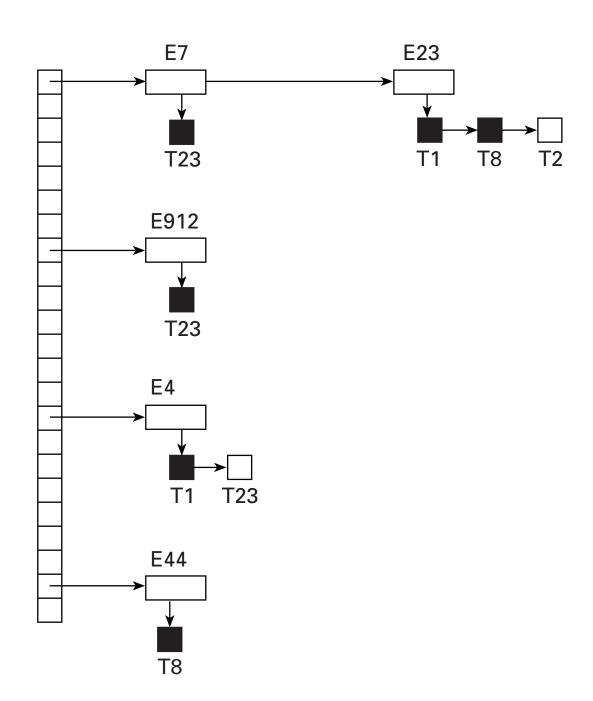


FIGURA 16.10. Tabla de bloqueos.

El gestor de bloqueos procesa las solicitudes de la siguiente forma:

- Cuando llega un mensaje de solicitud, añade un registro al final de la lista enlazada del elemento de datos, si la lista enlazada existe. En otro caso crea una nueva lista enlazada que tan sólo contiene el registro correspondiente a la solicitud.

  Siempre concede la primera solicitud de bloqueo sobre el elemento de datos. Pero si la transacción solicita un bloqueo sobre un elemento sobre el cual ya se ha concedido un bloqueo, el gestor de bloqueos sólo concede la solicitud si es compatible con las solicitudes anteriores, y todas éstas ya se han concedido. En otro caso, la solicitud tiene que esperar.
- Cuando el gestor de bloqueos recibe un mensaje de desbloqueo de una transacción, borra el registro para ese elemento de datos de la lista enlazada correspondiente a dicha transacción. Prueba el siguiente registro, si lo hay, como se describe en el párrafo anterior, para determinar si ahora se puede conceder dicha solicitud. Si se puede, el gestor de bloqueos concede la solicitud y procesa el siguiente registro, si lo hay, de forma similar, y así sucesivamente.
- Si una transacción se interrumpe, el gestor de bloqueos borra cualquier solicitud en espera realizada por la transacción. Una vez que el sistema de base de datos ha realizado las acciones apropiadas para deshacer la transacción (véase el Apartado 17.3), libera todos los bloqueos que mantenía la transacción abortada.

Este algoritmo garantiza que las solicitudes de bloqueo están libres de inanición, dado que una solicitud nunca se puede conceder mientras no se hayan concedido las solicitudes recibidas anteriormente. Más adelante, en el Apartado 16.6.3, se estudiará cómo detectar y manejar interbloqueos. El Apartado 18.2.1 describe una implementación alternativa —una que utiliza memoria compartida en vez de paso de mensajes para la solicitud y concesión de bloqueos.

16.1.5. Protocolos basados en grafos

Como se ha visto en el Apartado 16.1.3, en ausencia de información acerca de la forma en que se accede a los elementos de datos, el protocolo de bloqueo de dos fases es necesario y suficiente para asegurar la secuencialidad. Pero, si se desean desarrollar protocolos que no sean de dos fases, es necesario tener información adicional acerca de la forma en que cada transacción accede a la base de datos. Existen varios modelos que pueden ofrecer la información adicional, que difieren en la cantidad de información que proporcionan. El modelo más simple exige que se tenga un conocimiento previo acerca del orden en el cual se accede a los elementos de la base de datos. Dada esta información es posible construir protocolos de bloqueo que no sean de dos fases pero que no obstante aseguren la secuencialidad.

Para adquirir tal conocimiento previo, se impone un orden parcial $\rightarrow$ sobre el conjunto $\mathbf{D} = \{d_1, d_2, \ldots, d_n\}$ de todos los elementos de datos. Si $d_i \rightarrow d_j$ entonces toda transacción que acceda tanto a d_i como a d_j debe acceder a d_i antes de acceder a d_j. Este orden parcial puede ser el resultado de la organización tanto lógica como física de los datos, o se puede imponer únicamente con motivo del control de concurrencia.

El orden parcial implica que el conjunto **D** se pueda ver como un grafo dirigido acíclico denominado **grafo de la base de datos**. Para simplificar, este apartado centra su atención sólo en aquellos grafos que son árboles con raíz. Se va a mostrar un protocolo simple llamado *protocolo de árbol*, el cual está restringido a utilizar sólo bloqueos *exclusivos*. En las notas bibliográficas se proporcionan referencias a otros protocolos basados en grafos más complejos.

En el **protocolo de árbol** sólo se permite la instrucción de bloqueo bloquear-X. Cada transacción *Ti* puede bloquear un elemento de datos al menos una vez y debe seguir las siguientes reglas:

1. El primer bloqueo de T_i puede ser sobre cualquier elemento de datos.
2. Posteriormente, T_i puede bloquear un elemento de datos Q sólo si T_i está bloqueando actualmente al padre de Q.
3. Los elementos de datos bloqueados se pueden desbloquear en cualquier momento.
4. T_i no puede bloquear de nuevo un elemento de datos que ya haya bloqueado y desbloqueado anteriormente.

Todas las planificaciones que sean legales bajo el protocolo de árbol son secuenciables en cuanto a conflictos.

Para ilustrar este protocolo considérese el grafo de la base de datos de la Figura 16.11. Las cuatro transacciones siguientes siguen el protocolo de árbol sobre dicho grafo. Sólo se muestran las instrucciones de bloqueo y desbloqueo:

T_{10}: bloquear-X(B); bloquear-X(E); bloquear-X(D); desbloquear(B); desbloquear(E); bloquear-X(G); desbloquear(D); desbloquear(G).
T_{11}: bloquear-X(D); bloquear-X(H); desbloquear(D); desbloquear(H).
T_{12}: bloquear-X(B); bloquear-X(E); desbloquear(E); desbloquear(B).
T_{13}: bloquear-X(D); bloquear-X(H); desbloquear(D); desbloquear(H).

En la Figura 16.12 se describe una planificación posible en la que participan estas cuatro transacciones. Nótese que durante su ejecución, la transacción T_{10} realiza bloqueos sobre dos subárboles *disjuntos*.

Obsérvese que la planificación de la Figura 16.12 es secuenciable en cuanto a conflictos. Se puede demostrar que el protocolo de árbol no sólo asegura la secuencialidad en cuanto a conflictos, sino que también asegura la ausencia de interbloqueos.

El protocolo de árbol de la Figura 16.12 no asegura la recuperabilidad y la ausencia de cascadas. Para asegurar la recuperabilidad y la ausencia de cascadas, el protocolo se puede modificar para que no permita liberar bloqueos exclusivos hasta el final de la transacción. Mantener los bloqueos exclusivos hasta el final de la transacción reduce la concurrencia. Existe una alterna-

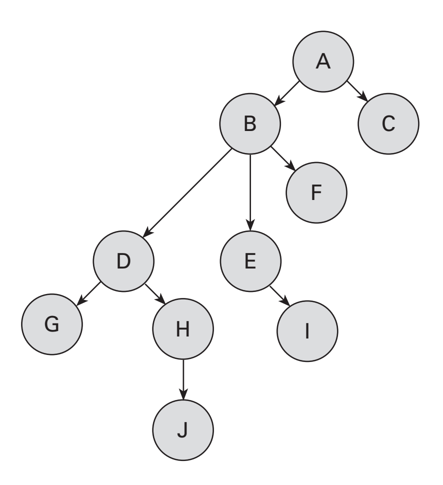


FIGURA 16.11. Grafo de la base de datos con estructura de árbol.

T_{10}	T_{11}	T_{12}	T_{13}
bloquear-X(B)			
	bloquear-X(D) bloquear-X(H) desbloquear(D)		
bloquear-X(E) bloquear-X(D) desbloquear(B) desbloquear(E)			
		bloquear-X(B) bloquear-X(E)	
	desbloquear(H)		
bloquear-X(G) desbloquear(D)			
			bloquear-X(D) bloquear-X(H) desbloquear(D) desbloquear(H)
		desbloquear(E) desbloquear(B)	
desbloquear(G)			

FIGURA 16.12. Planificación secuenciable bajo el protocolo de árbol.

tiva que mejora la concurrencia, pero sólo asegura la recuperabilidad: para cada elemento de datos con una escritura no comprometida se registra qué transacción realizó la última escritura sobre el elemento de datos. Cada vez que una transacción T_i realiza una lectura de un elemento de datos no comprometido, se registra una **dependencia de compromiso** de T_i en la transacción que realizó la última escritura sobre el elemento de datos. Entonces, no se permite que la transacción T_i se comprometa hasta que todas las transacciones sobre las que tiene una dependencia de compromiso se comprometan. Si alguna de estas transacciones se interrumpe, también hay que interrumpir T_i.

El protocolo de bloqueo de árbol tiene la ventaja sobre el protocolo de bloqueo de dos fases de que, a diferencia del bloqueo de dos fases, está libre de interbloqueos, así que no se necesitan retrocesos. El protocolo de bloqueo de árbol tiene otra ventaja sobre el protocolo de bloqueo de dos fases: que los desbloqueos se pueden producir antes. El hecho de desbloquear antes puede llevar a unos tiempos de espera menores y a un aumento de la concurrencia.

Sin embargo, el protocolo tiene el inconveniente de que, en algunos casos, una transacción puede que tenga que bloquear elementos de datos a los que no accede. Por ejemplo, una transacción que tenga que acceder a los elementos de datos A y J en el grafo de la base de datos de la Figura 16.11, debe bloquear no sólo A y J sino también los elementos de datos B, D y H. Estos bloqueos adicionales producen un aumento del coste de los bloqueos, la posibilidad de tiempos de espera adicionales y un descenso potencial de la concurrencia. Además, sin un conocimiento previo de los elementos de datos que es necesario bloquear, las transacciones tienen que bloquear la raíz del árbol y esto puede reducir considerablemente la concurrencia.

Para un conjunto de transacciones pueden existir planificaciones secuenciables en cuanto a conflictos que no se pueden obtener por medio del protocolo de árbol. Realmente hay planificaciones que son posibles por medio del protocolo de bloqueo de dos fases que no son posibles por medio del protocolo de árbol y viceversa. En los ejercicios se exponen ejemplos de tales planificaciones.

16.2. PROTOCOLOS BASADOS EN MARCAS TEMPORALES

En los protocolos de bloqueo que se han descrito antes se determina el orden entre dos transacciones conflictivas en tiempo de ejecución a través del primer bloqueo que soliciten ambas que traiga consigo modos incompatibles. Otro método para determinar el orden de secuencialidad es seleccionar previamente un orden entre las transacciones. El método más común para hacer esto es utilizar un esquema de *ordenación por marcas temporales*.

16.2.1. Marcas temporales

A toda transacción T_i del sistema se le asocia una única marca temporal fijada, denotada por MT(T_i). El sistema de base de datos asigna esta marca temporal antes de que comience la ejecución de T_i. Si a la transacción T_i se le ha asignado la marca temporal MT(T_i) y una nueva transacción T_j entra en el sistema, entonces MT(T_i) < MT(T_j). Existen dos métodos simples para implementar este esquema:

1. Usar el valor del *reloj del sistema* como marca temporal; es decir, la marca temporal de una transacción es igual al valor del reloj en el momento en el que la transacción entra en el sistema.
2. Usar un **contador lógico** que se incrementa cada vez que se asigna una nueva marca temporal; es decir, la marca temporal de una transacción es igual al valor del contador en el momento en el cual la transacción entra en el sistema.

Las marcas temporales de las transacciones determinan el orden de secuencia. De este modo, si MT(T_i) < MT(T_j) entonces el sistema debe asegurar que toda planificación que produzca es equivalente a una planificación secuencial en la cual la transacción T_i aparece antes que la transacción T_j.

Para implementar este esquema se asocia a cada elemento de datos Q dos valores de marca temporal:

- **marca_temporal-E**(Q) denota la mayor marca temporal de todas las transacciones que ejecutan con éxito escribir(Q).
- **marca_temporal-L**(Q) denota la mayor marca temporal de todas las transacciones que ejecutan con éxito leer(Q).

Estas marcas temporales se actualizan cada vez que se ejecuta una nueva operación leer(Q) o escribir(Q).

16.2.2. Protocolo de ordenación por marcas temporales

El **protocolo de ordenación por marcas temporales** asegura que todas las operaciones leer y escribir conflictivas se ejecutan en el orden de las marcas temporales. Este protocolo opera como sigue:

1. Supóngase que la transacción T_i ejecuta leer(Q).
   - **a.** Si MT(T_i) < marca_temporal-E(Q) entonces T_i necesita leer un valor de Q que ya se ha sobrescrito. Por tanto se rechaza la operación leer y T_i se retrocede.
   - **b.** Si MT(T_i) ≥ marca_temporal-E(Q) entonces se ejecuta la operación leer y marca_temporal-L(Q) se asigna al máximo de marca_temporal-L(Q) y de MT(T_i).
2. Supóngase que la transacción T_i ejecuta escribir(Q).
   - **a.** Si MT(T_i) < marca_temporal-L(Q) entonces el valor de Q que produce T_i se necesita previamente y el sistema asume que dicho valor no se puede producir nunca. Por tanto, se rechaza la operación escribir y T_i se retrocede.
   - **b.** Si MT(T_i) < marca_temporal-E(Q) entonces T_i está intentando escribir un valor de Q obsoleto. Por tanto, se rechaza la operación escribir y T_i se retrocede.
   - **c.** En otro caso se ejecuta la operación escribir y MT(T_i) se asigna a marca_temporal-E(Q).

A una transacción T_i que el esquema de control de concurrencia haya retrocedido como resultado de la ejecución de una operación leer o escribir se le asigna una nueva marca temporal y se inicia de nuevo.

Para ilustrar este protocolo considérense las transacciones T_{14} y T_{15}. La transacción T_{14} visualiza el contenido de las cuentas A y B:

```
T14: leer(B);
     leer(A);
     visualizar(A + B).
```

La transacción T_{15} transfiere 50 € de la cuenta A a la B y muestra después el contenido de ambas:

```
T15: leer(B);
     B := B – 50;
     escribir(B);
     leer(A);
     A := A + 50;
     escribir(A);
     visualizar(A + B).
```

En las planificaciones actuales con el protocolo de marcas temporales se asume que a una transacción se le asigna una marca temporal inmediatamente antes de su primera instrucción. Así en la planificación 3 de la Figura 16.13, MT(T_{14}) < MT(T_{15}) y la planificación con el protocolo de marcas temporales es posible.

Nótese que la ejecución anterior puede obtenerse también con el protocolo de bloqueo de dos fases. Sin embargo, existen planificaciones que son posibles con el protocolo de bloqueo de dos fases que no lo son con el protocolo de marcas temporales y viceversa (véase el Ejercicio 16.20).

El protocolo de ordenación por marcas temporales asegura la secuencialidad en cuanto a conflictos. Esta afirmación se deduce del hecho de que las operaciones conflictivas se procesan durante la ordenación de las marcas temporales.

El protocolo asegura la ausencia de interbloqueos, ya que ninguna transacción tiene que esperar.

Sin embargo, existe una posibilidad de inanición de las transacciones largas si una secuencia de transaccio-

T_{14}	T_{15}
leer(B)	
	leer(B) $B := B - 50$ escribir(B)
leer(A)	
	leer(A)
visualizar($A + B$)	
	$A := A + 50$ escribir(A) visualizar($A + B$)

FIGURA 16.13. Planificación 3.

T_{16}	T_{17}
leer(Q)	
	escribir(Q)
escribir(Q)	

FIGURA 16.14. Planificación 4.

nes cortas conflictivas provoca reinicios repetidos de la transacción larga. Si se descubre que una transacción se está reiniciando de forma repetida, es necesario bloquear las transacciones conflictivas de forma temporal para permitir que la transacción termine.

El protocolo puede generar planificaciones no recuperables. Sin embargo, se puede extender para producir planificaciones recuperables de una de las siguientes formas:

- La recuperabilidad y la ausencia de cascadas se pueden asegurar realizando todas las escrituras juntas al final de la transacción. Las escrituras tienen que ser atómicas en el siguiente sentido: mientras que las escrituras están en progreso, no se permite que ninguna transacción acceda a ninguno de los elementos de datos que se han escrito.
- La recuperabilidad y la ausencia de cascadas también se pueden garantizar utilizando una forma limitada de bloqueo, por medio de la cual las lecturas de los elementos no comprometidos se posponen hasta que la transacción haya actualizado el elemento comprometido (véase el Ejercicio 16.22).
- La recuperabilidad sola se puede asegurar realizando un seguimiento de las escrituras no comprometidas y sólo permitiendo que una transacción T_i se comprometa después de que se comprometa cualquier transacción que escribió un valor que leyó T_i. Las dependencias de compromiso, esbozadas en el Apartado 16.1.5, se pueden utilizar para este propósito.

16.2.3. Regla de escritura de Thomas

Ahora se presenta una modificación del protocolo de ordenación por marcas temporales que permite una mayor concurrencia potencial que la que tiene el protocolo del Apartado 16.2.2. Considérese la planificación 4 de la Figura 16.14 y aplíquese el protocolo de ordenación por marcas temporales. Puesto que T_{16} comienza antes que T_{17} se asume que MT(T_{16}) < MT(T_{17}). La operación leer(Q) de T_{16} tiene éxito, así como la operación escribir(Q) de T_{17}. Cuando T_{16} intenta hacer su operación escribir(Q) se encuentra con que MT(T_{16}) < marca_temporal-E(Q), ya que marca_temporal-E(Q) = MT(T_{17}). De este modo se rechaza la operación escribir(Q) de T_{16} y se debe retroceder la transacción.

A pesar de que el protocolo de ordenación por marcas temporales exige el retroceso de la transacción T_{16}, esto no es necesario. Como T_{17} ya ha escrito Q, el valor que intenta escribir T_{16} nunca se va a leer. Toda transacción T_i con MT(T_i) < MT(T_{17}) que intente una operación leer(Q) se retrocederá, ya que MT(T_i) < marca_temporal-E(Q). Toda transacción T_j con MT(T_j) > MT(T_{17}) debe leer el valor de Q que ha escrito T_{17} en lugar del valor que ha escrito T_{16}.

Esta observación lleva a la versión modificada del protocolo de ordenación por marcas temporales en el cual se pueden ignorar las operaciones escribir obsoletas bajo ciertas circunstancias. Las reglas del protocolo para las operaciones leer no sufren cambios. Sin embargo, las reglas del protocolo para las operaciones escribir son algo diferentes de las del protocolo de ordenación por marcas temporales del Apartado 16.2.2.

La modificación al protocolo de ordenación por marcas temporales, denominada **regla de escritura de Thomas**, es la siguiente. Supóngase que la transacción T_i ejecuta escribir(Q).

1. Si MT(T_i) < marca_temporal-L(Q), entonces el valor de Q que produce T_i se necesita previamente y el sistema asume que dicho valor no se puede producir nunca. Por tanto, se rechaza la operación escribir y T_i se retrocede.
2. Si MT(T_i) < marca_temporal-E(Q), entonces T_i está intentando escribir un valor de Q obsoleto. Por tanto se puede ignorar dicha operación escribir.
3. En otro caso, se ejecuta la operación escribir y MT(T_i) se asigna a marca_temporal-E(Q).

La diferencia entre las reglas anteriores y las del Apartado 16.2.2 está en la segunda regla. El protocolo de ordenación por marcas temporales exige que T_i se retroceda si ejecuta escribir(Q) y MT(T_i) < marca_temporal-E(Q). Sin embargo, ahora se ignora la instrucción escribir obsoleta en aquellos casos en los que MT(T_i) ≥ marca_temporal-L(Q).

La regla de escritura de Thomas hace uso de la secuencialidad en cuanto a vistas borrando las operaciones escribir obsoletas de las transacciones que las

ejecutan. Esta modificación de las transacciones hace posible generar planificaciones que no serían posibles con otros protocolos que se han presentado en este capítulo. Por ejemplo, la planificación 4 de la Figura 16.14 no es secuenciable en cuanto a conflictos y, por tanto, no es posible con cualquiera de los dos bloqueos de dos fases, con el protocolo de árbol o con el protocolo de ordenación por marcas temporales. Con la regla de escritura de Thomas se ignoraría la operación escribir(Q) de T_{16}. El resultado es una planificación que es equivalente en cuanto a vistas a la planificación secuencial $<T_{16},T_{17}>$.

16.3. PROTOCOLOS BASADOS EN VALIDACIÓN

En aquellos casos en que la mayoría de las transacciones son de sólo lectura, la tasa de conflictos entre las transacciones puede ser baja. Así, muchas de esas transacciones, si se ejecutasen sin la supervisión de un esquema de control de concurrencia, llevarían no obstante al sistema a un estado consistente. Un esquema de control de concurrencia impone una sobrecarga en la ejecución de código y un posible retardo en las transacciones. Sería aconsejable utilizar otro esquema alternativo que impusiera menos sobrecarga. La dificultad de reducir la sobrecarga está en que no se conocen de antemano las transacciones que estarán involucradas en un conflicto. Para obtener dicho conocimiento se necesita un esquema para **supervisar** el sistema.

Se asume que cada transacción T_i se ejecuta en dos o tres fases diferentes durante su tiempo de vida dependiendo de si es una transacción de sólo lectura o una de actualización. Las fases son, en orden,

1. **Fase de lectura.** Durante esta fase tiene lugar la ejecución de la transacción T_i. Se leen los valores de varios elementos de datos y se almacenan en variables locales de T_i. Todas las operaciones escribir se realizan sobre las variables locales temporales sin actualizar la base de datos actual.
2. **Fase de validación.** La transacción T_i realiza una prueba de validación para determinar si puede copiar a la base de datos las variables locales temporales que tienen los resultados de las operaciones escribir sin causar una violación de la secuencialidad.
3. **Fase de escritura.** Si la transacción T_i tiene éxito en la validación (paso 2) entonces las actualizaciones reales se aplican a la base de datos. En otro caso, T_i se retrocede.

Cada transacción debe pasar por las tres fases y en el orden que se muestra. Sin embargo, se pueden entrelazar las tres fases de la ejecución concurrente de las transacciones.

Para realizar la prueba de validación se necesita conocer el momento en que tienen lugar las distintas fases de las transacciones T_i. Se asociarán por tanto tres marcas temporales distintas a la transacción T_i:

1. **Inicio**(T_i), momento en el cual T_i comienza su ejecución.
2. **Validación**(T_i), momento en el cual T_i termina su fase de lectura y comienza su fase de validación.
3. **Fin**(T_i), momento en el cual T_i termina su fase de escritura.

Se determina el orden de secuencialidad a través de la técnica de la ordenación por marcas temporales utilizando el valor de la marca temporal Validación(T_i). De este modo, el valor MT(T_i) = Validación(T_i) y, si MT(T_j) < MT(T_k), entonces toda planificación que se produzca debe ser equivalente a una planificación secuencial en la cual la transacción T_j aparezca antes que la transacción T_k. La razón por la que se ha elegido Validación(T_i) como marca temporal de la transacción T_i, en lugar de Inicio(T_i), es porque se puede esperar un tiempo de respuesta más rápido, dado que la tasa de conflictos entre transacciones es realmente bajo.

La **comprobación de validación** para la transacción T_j exige que, para toda transacción T_i con MT(T_i) < MT(T_j), se cumplan una de las dos condiciones siguientes:

1. Fin(T_i) < Inicio(T_j). Puesto que T_i completa su ejecución antes de que comience T_j, el orden de secuencialidad se mantiene realmente.
2. El conjunto de todos los elementos de datos que escribe T_i tiene intersección vacía con el conjunto de elementos de datos que lee T_j, y T_i completa su fase de escritura antes de que T_j comience su fase de validación (Inicio(T_j) < Fin(T_i) < Validación(T_j)). Esta condición asegura que las escrituras de T_i y T_j no se superpongan. Puesto que las escrituras de T_i no afectan a la lectura de T_j, y puesto que T_j no puede afectar a la lectura de T_i, el orden de secuencialidad se mantiene realmente.

Como ejemplo considérense de nuevo las transacciones T_{14} y T_{15}. Supóngase que MT(T_{14}) < MT(T_{15}). Entonces la fase de validación tiene éxito y produce la planificación 5, la cual se describe en la Figura 16.15. Nótese que las escrituras a las variables actuales se realizan sólo después de la fase de validación de T_{15}. Así,

T_{14}	T_{15}
leer(B)	
	leer(B)
	$B := B - 50$
	leer(A)
	$A := A + 50$
leer(A)	
$\langle$ *validar* $\rangle$	
visualizar($A + B$)	
	$\langle$ *validar* $\rangle$
	escribir(B)
	escribir(A)

FIGURA 16.15. Planificación 5 producida usando validación.

T_{14} lee los valores anteriores de B y A y la planificación es secuenciable.

El esquema de validación evita automáticamente los retrocesos en cascada, ya que las escrituras reales tienen lugar sólo después de que la transacción que ejecuta la escritura se haya comprometido. Sin embargo, existe una posibilidad de inanición de las transacciones largas debido a una secuencia de transacciones cortas conflictivas que provoca reinicios repetidos de la transacción larga. Para evitar la inanición es necesario bloquear las transacciones conflictivas de forma temporal para permitir que la transacción larga termine.

Este esquema de validación se denomina esquema **control de concurrencia optimista**, dado que las transacciones se ejecutan de forma optimista, asumiendo que serán capaces de finalizar la ejecución y validar al final. En cambio, los bloqueos y la ordenación por marcas temporales son pesimistas, debido a que fuerzan una espera o un retroceso cada vez que se detecta un conflicto, aún cuando existe una posibilidad de que la planificación sea secuenciable en cuanto a conflictos.

16.4. GRANULARIDAD MÚLTIPLE

En los esquemas de control de concurrencia que se han descrito antes se ha tomado cada elemento de datos individual como la unidad sobre la cual se producía la sincronización.

Sin embargo, hay circunstancias en las que puede ser conveniente agrupar varios elementos de datos y tratarlos como una unidad individual de sincronización. Por ejemplo, si la transacción T_i tiene que acceder a toda la base de datos y se usa un protocolo de bloqueo, entonces T_i debe bloquear cada elemento de la base de datos. Ejecutar estos bloqueos produce claramente un consumo de tiempo. Sería mejor que T_i pudiera realizar una *única* petición de bloqueo para bloquear toda la base de datos. Por otro lado, si la transacción T_j necesita acceder sólo a unos cuantos datos no sería necesario bloquear toda la base de datos, ya que en ese caso se perdería la concurrencia.

Lo que se necesita es un mecanismo que permita al sistema definir múltiples niveles de **granularidad**. Se puede hacer uno permitiendo que los elementos de datos sean de varios tamaños y definiendo una jerarquía de granularidades de los datos, en la cual las granularidades pequeñas están anidadas en otras más grandes. Una jerarquía tal se puede representar gráficamente como un árbol. Nótese que el árbol que se usa aquí es significativamente distinto del que se usaba en el protocolo de árbol (Apartado 16.1.5). Un nodo interno del árbol de granularidad múltiple representa los datos que se asocian con sus descendientes. En el protocolo de árbol cada nodo es un elemento de datos independiente.

Como ejemplo considérese el árbol de la Figura 16.16, el cual consiste en cuatro niveles de nodos. El nivel superior representa toda la base de datos. Después de éste están los nodos de tipo *zona*; la base de datos consiste en estas zonas exactamente. Cada zona tiene nodos de tipo *archivo* como hijos. Cada zona contiene exactamente aquellos archivos que sean sus nodos hijos. Ningún archivo está en más de una zona. Finalmente cada archivo tiene nodos de tipo *registro*. Como antes, un archivo consiste exactamente en aquellos registros que sean sus nodos hijos y ningún registro puede estar presente en más de un archivo.

Se puede bloquear individualmente cada nodo del árbol. Como ya se hizo en el protocolo de bloqueo de dos fases, se van a usar los modos de bloqueo **compartido** y **exclusivo**. Cuando una transacción bloquea un nodo, tanto en modo compartido como exclusivo, también bloquea todos los descendientes de ese nodo con el mismo modo de bloqueo. Por ejemplo, si la transacción T_i **bloquea explícitamente** el archivo A_c de la Figura 16.16 en modo exclusivo, entonces **bloquea implícitamente** en modo exclusivo todos los registros que pertenecen a dicho archivo. No es necesario que bloquee explícitamente cada registro individual de A_c.

Supóngase que la transacción T_j quiere bloquear el registro r_{b_6} del archivo A_b. Puesto que *Ti* ha bloqueado A_b explícitamente, se entiende que también se bloquea r_{b_6} (implícitamente). Pero cuando T_j realiza una petición de bloqueo sobre r_{b_6}, ¡r_{b_6} no está bloqueado explícitamente! ¿Cómo puede determinar el sistema si T_j puede bloquear r_{b_6}? T_j debe recorrer el árbol desde la raíz hasta el nodo r_{b_6}. Si cualquier nodo del camino está bloqueado en un modo incompatible entonces hay que retrasar T_j.

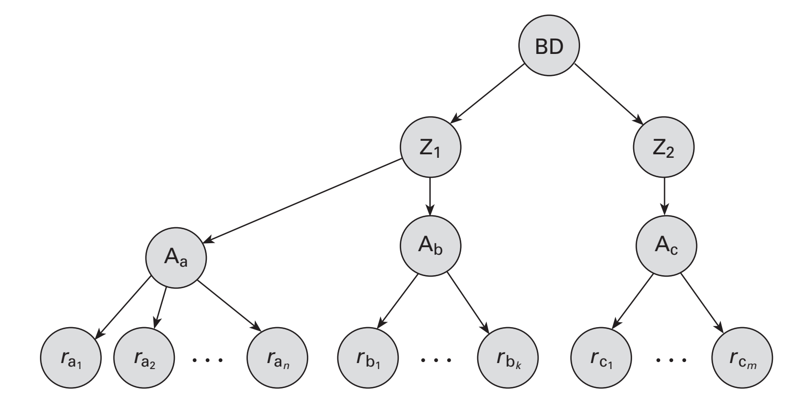


FIGURA 16.16. Jerarquía de granularidad.

Supóngase ahora que la transacción T_k quiere bloquear toda la base de datos. Para hacer esto debe bloquear simplemente la raíz de la jerarquía. Nótese, sin embargo, que T_k no tendrá éxito al bloquear el nodo raíz, ya que T_i posee un bloqueo sobre parte del árbol (en concreto sobre el archivo A_b). ¿Pero cómo determina el sistema si se puede bloquear el nodo raíz? Una posibilidad es buscar en todo el árbol. Sin embargo esta solución anula el propósito del esquema de bloqueo de granularidad múltiple. Un modo más eficiente de obtener este conocimiento es introducir una nueva clase de modo de bloqueo que se denomina **modo de bloqueo intencional**. Si un nodo se bloquea en modo intencional se está haciendo un bloqueo explícito en un nivel inferior del árbol (es decir, en una granularidad más fina). Los bloqueos intencionales se colocan en todos los ascendientes de un nodo antes de bloquearlo explícitamente. Así, no es necesario que una transacción busque en todo el árbol para determinar si puede bloquear un nodo con éxito. Una transacción que quiera bloquear un nodo —por ejemplo Q— debe recorrer el camino en el árbol desde la raíz hasta Q. Durante el recorrido del árbol la transacción bloquea los distintos nodos en un modo intencional.

Hay un modo intencional asociado al modo compartido y hay otro asociado al modo exclusivo. Si un nodo se bloquea en **modo intencional-compartido (IC)** se realiza un bloqueo explícito en un nivel inferior del árbol, pero sólo con bloqueos en modo compartido. De forma similar, si se bloquea un nodo en **modo intencional-exclusivo (IX)** entonces el bloqueo explícito se hace en un nivel inferior con bloqueos en modo exclusivo o en modo compartido. Finalmente, si se bloquea un nodo en **modo intencional-exclusivo y compartido (IXC)** el subárbol cuya raíz es ese nodo se bloquea explícitamente en modo exclusivo, y dicho bloqueo explícito se produce en un nivel inferior con bloqueos en modo exclusivo. La función de compatibilidad para estos modos de bloqueo se presenta en la Figura 16.17.

El **protocolo de bloqueo de granularidad múltiple** siguiente asegura la secuencialidad. Cada transacción T_i puede bloquear un nodo Q usando las reglas siguientes:

1. Debe observar la función de compatibilidad de bloqueos de la Figura 16.17.
2. Debe bloquear la raíz del árbol en primer lugar y puede bloquearla en cualquier modo.
3. Puede bloquear un nodo Q en modo C o IC sólo si está bloqueando actualmente al padre de Q en modo IX o IC.
4. Puede bloquear un nodo Q en modo X, IXC o IX sólo si está bloqueando actualmente al padre de Q en modo IX o IXC.
5. Puede bloquear un nodo sólo si no ha desbloqueado previamente ningún nodo (es decir, T_i es de dos fases).
6. Puede desbloquear un nodo Q sólo si no ha bloqueado a ninguno de los hijos de Q.

Obsérvese que en el protocolo de granularidad múltiple es necesario que se adquieran los bloqueos en *orden descendente* (de la raíz a las hojas), y que se liberen en *orden ascendente* (de las hojas a la raíz).

Para ilustrar este protocolo considérese el árbol de la Figura 16.16 y las transacciones siguientes:

- Supóngase que la transacción T_{18} lee el registro r_{a_2} del archivo A_a. Entonces T_{18} necesita bloquear la

	IC	IX	C	IXC	X
IC	cierto	cierto	cierto	cierto	falso
IX	cierto	cierto	falso	falso	falso
C	cierto	falso	cierto	falso	falso
IXC	cierto	falso	falso	falso	falso
X	falso	falso	falso	falso	falso

FIGURA 16.17. Matriz de compatibilidad.

base de datos, la zona Z_1 y A_a en modo IC (y en este orden) y finalmente debe bloquear r_{a_2} en modo C.

- Supóngase que la transacción T_{19} modifica el registro r_{a_9} del archivo A_a. Entonces T_{19} necesita bloquear la base de datos, la zona Z_1 y el archivo A_a en modo IX y finalmente debe bloquear r_{a_9} en modo X.
- Supóngase que la transacción T_{20} lee todos los registros del archivo A_a. Entonces, T_{20} necesita bloquear la base de datos y la zona Z_1 (y ello en este orden) en modo IC y finalmente debe bloquear A_a en modo C.
- Supóngase que la transacción T_{21} lee toda la base de datos. Puede hacer esto una vez que bloquee la base de datos en modo C.

Se observa que las transacciones T_{18}, T_{20} y T_{21} pueden acceder concurrentemente a la base de datos. La transacción T_{19} se puede ejecutar concurrentemente con T_{18}, pero no con T_{20} ni con T_{21}.

Este protocolo permite la concurrencia y reduce la sobrecarga de bloqueos. Es particularmente útil en las aplicaciones con una mezcla de

- Transacciones cortas que sólo acceden a algunos elementos de datos.
- Transacciones largas que producen informes a partir de un archivo completo o un conjunto de archivos.

Existe un protocolo de bloqueo similar que es aplicable a sistemas de bases de datos en los cuales las granularidades de los datos se organizan en forma de grafo dirigido acíclico. Véanse las notas bibliográficas para obtener referencias adicionales. En este protocolo son posibles los interbloqueos, al igual que en los protocolos de bloqueo de dos fases. Existen técnicas para reducir la frecuencia de interbloqueos en el protocolo de granularidad múltiple y también para eliminar los interbloqueos completamente. Se hace referencia a estas técnicas en las notas bibliográficas.

16.5. ESQUEMAS MULTIVERSIÓN

Los esquemas de control de concurrencia que se han descrito anteriormente aseguran la secuencialidad o bien retrasando una operación o bien abortando la transacción que realiza la operación. Por ejemplo, una operación leer se puede retrasar porque todavía no se haya escrito el valor apropiado; o se puede rechazar (es decir, la transacción que la realiza debe abortarse) porque se haya sobrescrito el valor que supuestamente se iba a leer. Se podrían evitar estos problemas si se mantuvieran en el sistema copias anteriores de cada elemento de datos.

En los esquemas de **control de concurrencia multiversión**, cada operación escribir(Q) crea una nueva **versión** de Q. Cuando se realiza una operación leer(Q) el gestor de control de concurrencia elige una de las versiones de Q que se va a leer. El esquema de control de concurrencia debe asegurar que la elección de la versión que se va a leer se haga de tal manera que asegure la secuencialidad. Así mismo es crucial, por motivos de rendimiento, que una transacción sea capaz de determinar rápida y fácilmente la versión del elemento de datos que se va a leer.

16.5.1. Ordenación por marcas temporales multiversión

La técnica más frecuentemente utilizada en los esquemas multiversión es la de las marcas temporales. A cada transacción T_i del sistema se le asocia una única marca temporal estática que se denota MT(T_i). El sistema de bases de datos asigna dicha marca temporal antes de que la transacción comience su ejecución, como se ha descrito en el Apartado 16.2.

A cada elemento de datos Q se el asocia una secuencia de versiones <Q_1, Q_2, … , Q_m>. Cada versión Q_k contiene tres campos:

- **contenido** es el valor de la versión Q_k.
- **marca_temporal-E**(Q_k) es la marca temporal de la transacción que haya creado la versión Q_k.
- **marca_temporal-L**(Q_k) es la mayor marca temporal de todas las transacciones que hayan leído con éxito la versión Q_k.

Una transacción —por ejemplo, T_i— crea una nueva versión Q_k del elemento de datos Q realizando la operación escribir(Q). El campo contenido de la versión tiene el valor que ha escrito T_i. El sistema inicializa la marca_temporal-E y marca_temporal-L con MT(T_i). El valor marca_temporal-L se actualiza cada vez que una transacción T_j lee el contenido de Q_k y marca_temporal-L(Q_k) < MT(T_j).

El **esquema de marcas temporales multiversión** que se muestra a continuación asegura la secuencialidad. El esquema opera como sigue. Supóngase que la transacción T_i realiza una operación leer(Q) o escribir(Q). Sea Q_k la versión de Q cuya marca temporal de escritura es la mayor marca temporal menor o igual que MT(T_i).

1. Si la transacción T_i ejecuta leer(Q), entonces el valor que se devuelve es el contenido de la versión Q_k.
2. Si la transacción T_i ejecuta escribir(Q) y si MT(T_i) < marca_temporal-L(Q_k), entonces la transacción

T_i se retrocede. Si no, si MT(T_i) = marca_temporal-E(Q_k) se sobrescribe el contenido de Q_k, y en otro caso se crea una nueva versión de Q.

La justificación de la regla 1 es clara. Una transacción lee la versión más reciente que viene antes de ella en el tiempo. La segunda regla fuerza a que se aborte una transacción que realice una escritura «demasiado tarde». Con más precisión, si T_i intenta escribir una versión que alguna otra transacción haya leído, entonces no se puede permitir que dicha escritura tenga éxito.

Las versiones que ya no se necesitan se borran basándose en la regla siguiente. Supóngase que hay dos versiones, Q_k y Q_j, de un elemento de datos y que ambas tienen marca_temporal-E menor que la marca temporal de la transacción más antigua en el sistema. Entonces no se volverá a usar la versión más antigua de Q_k y Q_j y se puede borrar.

El esquema de ordenación por marcas temporales multiversión tiene la propiedad deseable de que nunca falla una petición de lectura y nunca tiene que esperar. En sistemas de bases de datos típicos, en los cuales la lectura es una operación mucho más frecuente que la escritura, esta ventaja puede tener un mayor significado práctico.

Este esquema, sin embargo, tiene dos propiedades no deseables. En primer lugar, la lectura de un elemento de datos necesita también la actualización del campo marca_temporal-L, lo que tiene como resultado dos accesos potenciales al disco en lugar de uno. En segundo lugar, los conflictos entre las transacciones se resuelven por medio de retrocesos en lugar de esperas. Esta alternativa puede ser costosa. En el Apartado 16.5.2 se describe un algoritmo que evita este problema.

Este esquema de ordenación por marcas temporales multiversión no asegura la recuperabilidad y la ausencia de cascadas. Se puede extender de la misma forma que el esquema de ordenación por marcas temporales básico, para hacerlo recuperable y sin cascadas.

16.5.2. Bloqueo de dos fases multiversión

El **protocolo de bloqueo de dos fases multiversión** intenta combinar las ventajas del control de concurrencia multiversión con las ventajas del bloqueo de dos fases. Este protocolo distingue entre **transacciones de sólo lectura** y **transacciones de actualización**.

Las transacciones de actualización realizan un bloqueo de dos fases riguroso; es decir, mantienen todos los bloqueos hasta el final de la transacción. Así, se pueden secuenciar según su orden de terminación. Cada elemento de datos tiene una única marca temporal. La marca temporal no es en este caso una marca temporal basada en un reloj real, sino que es un contador, que se denomina **contador_mt**, que se incrementa durante el procesamiento del compromiso.

A las transacciones de sólo lectura se les asigna una marca temporal leyendo el valor actual de **contador_mt** antes de que comiencen su ejecución; para realizar las lecturas siguen el protocolo de ordenación por marcas temporales multiversión. Así, cuando una transacción de sólo lectura T_i ejecuta leer(Q), el valor que se devuelve es el contenido de la versión con la mayor marca temporal menor que MT(T_i).

Cuando una transacción de actualización lee un elemento obtiene un bloqueo compartido sobre él y lee la versión más reciente de dicho elemento de datos. Cuando una transacción de actualización quiere escribir un elemento, obtiene primero un bloqueo exclusivo sobre el elemento, y luego crea una nueva versión del elemento de datos. La escritura se realiza sobre la nueva versión y la marca temporal de la nueva versión tiene inicialmente el valor ∞, el cual es mayor que el de cualquier marca temporal posible.

Una vez que la transacción T_i ha completado sus acciones realiza el procesamiento del compromiso como sigue: en primer lugar, T_i asigna la marca temporal de todas las versiones que ha creado al valor de **contador_mt** más 1; después T_i incrementa **contador_mt** en 1. Sólo se permite que realice el procesamiento del compromiso a una transacción de actualización a la vez.

Como resultado, las transacciones que comiencen después de que T_i incremente **contador_mt** observarán los valores que T_i ha actualizado, mientras que aquellas que comiencen antes de que T_i incremente **contador_mt** observarán los valores anteriores a la actualización de T_i. En ambos casos las transacciones de sólo lectura no tienen que esperar nunca por los bloqueos. El bloqueo de dos fases multiversión también asegura que las planificaciones son recuperables y sin cascadas.

Las versiones se borran de forma similar a la utilizada en la ordenación por marcas temporales multiversión. Supóngase que hay dos versiones, Q_k y Q_j, de un elemento de datos y que ambas versiones tienen una marca temporal menor que la de la transacción de sólo lectura más antigua del sistema. Entonces la versión más antigua de Q_k y Q_j no se volverá a utilizar y se puede borrar.

El bloqueo de dos fases multiversión o variaciones de éste se usan en algunos sistemas de bases de datos comerciales.

16.6. TRATAMIENTO DE INTERBLOQUEOS

Un sistema está en estado de interbloqueo si existe un conjunto de transacciones tal que toda transacción del conjunto está esperando a otra transacción del conjunto. Con mayor precisión, existe un conjunto de transacciones en espera $\{T_0, T_1, \ldots, T_n\}$ tal que T_0 está esperando a un elemento de datos que posee T_1, y T_2 está esperando a un elemento de datos que posee T_2, y …, y T_{n-1} está esperando a un elemento de datos que posee $T_{n'}$ y T_n está esperando a un elemento que posee T_0. En tal situación ninguna de las transacciones puede progresar.

El único remedio a esta situación no deseada es que el sistema invoque alguna acción drástica, como retroceder alguna de las transacciones involucradas en el interbloqueo. El retroceso de una transacción puede ser parcial: esto es, se puede retroceder una transacción hasta el punto donde obtuvo un bloqueo cuya liberación resuelve el interbloqueo.

Existen dos métodos principales para tratar el problema de los interbloqueos. Se puede utilizar un protocolo de **prevención de interbloqueos** para asegurar que el sistema *nunca* llega a un estado de interbloqueo. De forma alternativa se puede permitir que el sistema llegue a un estado de interbloqueo, y tratar de recuperarse después a través de un esquema de **detección y recuperación de interbloqueos**. Como se verá a continuación, ambos pueden provocar retroceso de las transacciones. La prevención se usa normalmente cuando la probabilidad de que el sistema llegue a un estado de interbloqueo es relativamente alta; en otro caso es más eficiente usar la detección y recuperación.

Nótese que el esquema de detección y recuperación necesita una sobrecarga que incluye no sólo el coste en tiempo de ejecución para mantener la información necesaria para ejecutar el algoritmo de detección, sino también las pérdidas potenciales inherentes a la recuperación de un interbloqueo.

16.6.1. Prevención de interbloqueos

Existen dos enfoques a la prevención de interbloqueos. Un enfoque asegura que no puede haber esperas cíclicas ordenando las peticiones de bloqueo o exigiendo que todos los bloqueos se adquieran juntos. El otro enfoque es más cercano a la recuperación de interbloqueos y realiza retrocesos de las transacciones en lugar de esperar un bloqueo, siempre que el bloqueo pueda llevar potencialmente a un interbloqueo.

El esquema más simple para la primera aproximación exige que cada transacción bloquee todos sus elementos de datos antes de comenzar su ejecución. Además, o bien se bloquean todos en un paso o ninguno de ellos se bloquea. Este protocolo tiene dos inconvenientes principales: 1) a menudo es difícil predecir, antes de que comience la transacción, cuáles son los elementos de datos que es necesario bloquear; 2) la utilización de elementos de datos puede ser muy baja, ya que muchos de los elementos de datos pueden estar bloqueados pero sin usar durante mucho tiempo.

Otro esquema para prevenir interbloqueos consiste en imponer un orden parcial a todos los elementos de datos y exigir que una transacción bloquee un elemento de datos sólo en el orden que especifica dicho orden parcial. Se ha visto un esquema así en el protocolo de árbol, que usa una ordenación parcial de los elementos de datos.

Una variante a esta aproximación es utilizar un orden total de los elementos de datos, en conjunción con el bloqueo de dos fases. Una vez que una transacción ha bloqueado un elemento en particular, no puede solicitar bloqueos sobre elementos que preceden a dicho elemento en la ordenación. Este esquema es fácil de implementar siempre que el conjunto de los elementos de datos a los que accede una transacción se conozca cuando la transacción comienza la ejecución. No hay necesidad de cambiar el sistema de control de concurrencia subyacente si se utiliza bloqueo de dos fases: todo lo que hace falta es asegurar que los bloqueos se solicitan en el orden adecuado.

La segunda aproximación para prevenir interbloqueos consiste en utilizar expropiación y retrocesos de transacciones. En la expropiación, cuando la transacción T_2 solicita un bloqueo que la transacción T_1 posee, el bloqueo concedido a T_1 debe **expropiarse** retrocediendo T_1 y concediéndolo a continuación a T_2. Para controlar la expropiación se asigna una marca temporal única a cada transacción. El sistema usa estas marcas temporales sólo para decidir si una transacción debe esperar o retroceder. Para el control de concurrencia se sigue usando el bloqueo. Si una transacción se retrocede mantiene su marca temporal *antigua* cuando vuelva a comenzar. Se han propuesto dos esquemas de prevención de interbloqueos que usan marcas temporales:

1. El esquema **esperar-morir** está basado en una técnica sin expropiación. Cuando la transacción T_i solicita un elemento de datos que posee actualmente T_j, T_i puede esperar sólo si tiene una marca temporal más pequeña que la de T_j (es decir, T_i es anterior a T_j). En otro caso T_i se retrocede (muere). Por ejemplo, supóngase que las transacciones T_{22}, T_{23} y T_{24} tienen marcas temporales 5, 10 y 15 respectivamente. Si T_{22} solicita un elemento de datos que posee T_{23} entonces T_{22} tiene que esperar. Si T_{24} solicita un elemento de datos que posee T_{23} entonces T_{24} se retrocede.

2. El esquema **herir-esperar** está basado en una técnica de expropiación. Es el opuesto del esquema esperar-morir. Cuando la transacción T_i solicita un elemento de datos que posee actualmente T_j, T_i pue-

de esperar sólo si tiene una marca temporal mayor que la de T_j (es decir, T_i es más reciente que T_j). En otro caso T_j se retrocede (T_i *hiere* a T_j). Volviendo al ejemplo anterior con las transacciones T_{22}, T_{23} y T_{24}, si T_{22} solicita un elemento de datos que posee T_{23} entonces se expropia este elemento de datos y T_{23} se retrocede. Si T_{24} solicita el elemento de datos que posee T_{23} entonces T_{24} debe esperar.

Siempre que se retrocedan las transacciones, es importante asegurarse de que no exista **inanición**, es decir, ninguna transacción se retrocede repetidamente y nunca se le permite progresar.

Tanto el esquema herir-esperar como el esperar-morir impiden la inanición: en todo momento hay una transacción con la menor marca temporal. Dicha transacción *no se puede* retroceder en ninguno de los dos esquemas. Puesto que las marcas temporales siempre se incrementan, y puesto que *no* se asignan a las transacciones nuevas marcas temporales cuando se retroceden, una transacción que se retrocede tendrá finalmente la menor marca temporal. De este modo no se retrocederá de nuevo.

Sin embargo, existen algunas diferencias en el modo en que operan estos dos esquemas.

- En el esquema esperar-morir una transacción más antigua debe esperar a que una más reciente libere sus elementos de datos. Así, cuanto más antigua se vuelve una transacción más tiende a esperar. Por el contrario, en el esquema herir-esperar, una transacción más antigua nunca espera a una más reciente.
- En el esquema esperar-morir, si la transacción T_i muere y se retrocede a causa de que ha solicitado un elemento de datos que posee la transacción T_j, entonces T_i puede volver a realizar la misma secuencia de peticiones cuando vuelva a comenzar. Si T_j posee todavía el elemento de datos, entonces T_i morirá de nuevo. De este modo T_i morirá varias veces hasta que adquiera el elemento de datos que necesita. Compárese esta secuencia de sucesos con la que ocurre en el esquema herir-esperar. La transacción T_i está herida y se retrocede porque T_j solicita un elemento de datos que posee. Cuando vuelve a comenzar T_i y solicita el elemento de datos que ahora posee T_j, T_i espera. De este modo puede haber menos retrocesos en el esquema herir-esperar.

El principal problema de cualquiera de estos dos esquemas es que se pueden dar retrocesos innecesarios.

16.6.2. Esquemas basados en límite de tiempo

Otro enfoque simple al tratamiento de interbloqueos se basa en **bloqueos con límite de tiempo**. En este enfoque una transacción que haya solicitado un bloqueo espera por lo menos durante un intervalo de tiempo especificado. Si no se concede el bloqueo en ese tiempo, se dice que la transacción está fuera de tiempo y ella misma se retrocede y vuelve a comenzar. Si de hecho había un interbloqueo, una o varias de las transacciones involucradas en dicho interbloqueo estarán fuera de tiempo y se retrocederán, lo que permitirá continuar a las demás. Este esquema se encuentra en un lugar entre la prevención de interbloqueos, donde nunca puede haber un interbloqueo, y la detección y recuperación, las cuales se describen en el Apartado 16.6.3.

El esquema de límite de tiempo es particularmente fácil de implementar y funciona bien si las transacciones son cortas y, si son largas, las esperas deben de ser a causa de los interbloqueos. Sin embargo, es difícil en general decidir cuánto debe esperar una transacción para que esté fuera de tiempo. Esperas demasiado largas provocan retardos innecesarios una vez que se ha producido un interbloqueo. Esperas demasiado cortas producen el retroceso de la transacción incluso si no hay interbloqueo, provocando un gasto de recursos. La inanición es también posible con este esquema. Por tanto el esquema basado en límite de tiempo tiene una aplicación limitada.

16.6.3. Detección y recuperación de interbloqueos

Si el sistema no utiliza algún protocolo que asegure la ausencia de interbloqueos, entonces se debe usar un esquema de detección y recuperación. Periódicamente se invoca a un algoritmo que examina el estado del sistema para determinar si hay un interbloqueo. Si lo hay, entonces el sistema debe intentar recuperarse del interbloqueo. Para ello, el sistema debe:

- Mantener información sobre la asignación de los elementos de datos a las transacciones, así como de toda petición de elemento de datos pendiente.
- Proporcionar un algoritmo que use esta información para determinar si el sistema ha entrado en un estado de interbloqueo.
- Recuperarse del interbloqueo cuando el algoritmo de detección determine que existe un interbloqueo.

En este apartado se van a desarrollar estos puntos.

16.6.3.1. Detección de interbloqueos

Los interbloqueos se pueden describir con precisión por medio de un grafo dirigido llamado **grafo de espera**. Este grafo consiste en un par $G = (V, A)$, siendo V el conjunto de vértices y A el conjunto de arcos. El conjunto de vértices consiste en todas las transacciones del sistema. Cada elemento del conjunto E de arcos es un par ordenado $T_i \rightarrow T_j$. Si $T_i \rightarrow T_j$ pertenece a A entonces hay un arco dirigido de la transacción T_i a T_j, lo cual implica que la transacción T_i está esperando a que la transacción T_j libere un elemento de datos que necesita.

Cuando la transacción T_i solicita un elemento de datos que posee actualmente la transacción T_j, entonces se inserta el arco $T_i \rightarrow T_j$ en el grafo de espera. Este arco sólo se borra cuando la transacción T_j deja de poseer un elemento de datos que necesite la transacción T_i.

Existe un interbloqueo en el sistema si y sólo si el grafo de espera contiene un ciclo. Se dice que toda transacción involucrada en el ciclo está interbloqueada. Para detectar los interbloqueos el sistema debe mantener el grafo de espera y debe invocar periódicamente a un algoritmo que busque un ciclo en el grafo.

Para ilustrar estos conceptos considérese el grafo de espera de la Figura 16.18, el cual describe la situación siguiente:

- La transacción T_{25} está esperando a las transacciones T_{26} y T_{27}.
- La transacción T_{27} está esperando a la transacción T_{26}.
- La transacción T_{26} está esperando a la transacción T_{28}.

Puesto que el grafo no tiene ciclos, el sistema no está en un estado de interbloqueo.

Supóngase ahora que la transacción T_{28} solicita un elemento que posee T_{27}. Se añade el arco $T_{28} \rightarrow T_{27}$ al grafo de espera, lo que resulta en el nuevo estado que se ilustra en la Figura 16.19. Ahora el grafo tiene el ciclo

$$T_{26} \rightarrow T_{28} \rightarrow T_{27} \rightarrow T_{26}$$

lo que implica que las transacciones T_{26}, T_{27} y T_{28} están interbloqueadas.

Por consiguiente surge la pregunta: ¿cuándo se debe invocar al algoritmo de detección? La respuesta depende de dos factores:

1. ¿Cada cuánto tiempo tiene lugar un interbloqueo?
2. ¿Cuántas transacciones se verán afectadas por el interbloqueo?

Si los interbloqueos ocurren con frecuencia, entonces se debe invocar al algoritmo de detección con mayor frecuencia de la usual. Los elementos de datos asignados a las transacciones interbloqueadas no estarán disponibles para otras transacciones hasta que pueda romperse el interbloqueo. Adicionalmente puede también aumentar

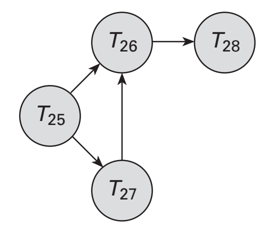


FIGURA 16.18. Grafo de espera sin ciclos.

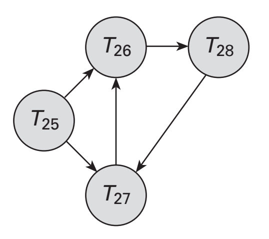


FIGURA 16.19. Grafo de espera con un ciclo.

el número de ciclos en el grafo. En el caso peor se invoca al algoritmo de detección cada vez que una petición de asignación no se pueda conceder inmediatamente.

16.6.3.2. Recuperación de interbloqueos

Cuando un algoritmo de detección de interbloqueos determina que existe un interbloqueo, el sistema debe **recuperarse** del mismo. La solución más común es retroceder una o más transacciones para romper el interbloqueo. Se deben realizar tres acciones:

1. **Selección de una víctima.** Dado un conjunto de transacciones interbloqueadas se debe determinar la transacción (o transacciones) que se van a retroceder para romper el interbloqueo. Se deben retroceder aquellas transacciones que incurran en un coste mínimo. Por desgracia, el término *coste mínimo* no es preciso. Hay muchos factores que determinan el coste de un retroceso, como
   - **a.** Lo que la transacción ha calculado y lo que se calculará hasta completar su tarea designada.
   - **b.** El número de elementos de datos que ha usado la transacción.
   - **c.** La cantidad de elementos de datos que necesita la transacción para que se complete.
   - **d.** El número de transacciones que se verán involucradas en el retroceso.
2. **Retroceso.** Una vez que se ha decidido que se retrocederá una transacción en particular, se debe determinar hasta dónde se retrocederá dicha transacción.

   La solución más simple consiste en un **retroceso total**: se aborta la transacción y luego vuelve a comenzar. Sin embargo, es más efectivo retroceder la transacción sólo lo necesario para romper el interbloqueo. Dicho **retroceso parcial** requiere que el sistema mantenga información adicional sobre el estado de todas las transacciones que están en ejecución. Concretamente, es necesario registrar la secuencia de solicitudes / concesiones de bloqueos y de actualizaciones realizadas por la transacción. El mecanismo de detección de interbloqueos debería decidir qué bloqueos necesita liberar la transacción seleccionada para romper el interbloqueo. Hay que retroceder la transacción seleccionada hasta el punto donde obtuvo el primero de esos bloqueos, deshaciendo todas las acciones que realizó

desde ese punto. El mecanismo de recuperación debe ser capaz de realizar tales retrocesos parciales. Además, las transacciones deben ser capaces de reanudar la ejecución después de un retroceso parcial. Véase el apartado de notas bibliográficas para obtener las referencias pertinentes.

3. **Inanición.** En un sistema en el cual la selección de víctimas esté basada principalmente en factores de coste, puede ocurrir que siempre se elija a la misma transacción como víctima. El resultado es que esta transacción no completa nunca su tarea designada. Dicha situación se denomina **inanición**. Se debe asegurar que una transacción puede elegirse como víctima sólo un número finito (y pequeño) de veces. La solución más común consiste en incluir en el factor de coste el número de retrocesos.

16.7. OPERACIONES PARA INSERTAR Y BORRAR

Hasta ahora se ha centrado la atención en las operaciones leer y escribir. Esta ligadura limita a las transacciones a los elementos de datos que ya están en la base de datos. Algunas transacciones necesitan no sólo acceder a los elementos de datos existentes, sino también poder crear nuevos elementos de datos. Otras necesitan tener la posibilidad de borrar elementos de datos. Para examinar la forma en que tales transacciones afectan al control de concurrencia se introducen las operaciones adicionales siguientes:

- **borrar**(Q) borra de la base de datos el elemento de datos Q.
- **insertar**(Q) inserta en la base de datos el nuevo elemento de datos Q y le asigna un valor inicial.

Si la transacción T_i intenta ejecutar una operación leer(Q) después de haberse borrado Q se produce un error lógico en T_i. Igualmente, si la transacción T_i intenta ejecutar una operación leer(Q) antes de que Q se haya insertado produce un error lógico en T_i. También es un error lógico intentar borrar un dato inexistente.

16.7.1. Borrado

Para comprender la manera en que puede afectar la presencia de las instrucciones **borrar** al control de concurrencia, se debe decidir en qué casos una instrucción **borrar** está en conflicto con otra instrucción. Sean I_i e I_j instrucciones de T_i y T_j, respectivamente, que están consecutivas en la planificación S. Sea I_i = **borrar**(Q). Se consideran distintas instrucciones I_j

- I_j = leer(Q). I_i e I_j están en conflicto. Si I_i está antes de I_j, T_j tendrá un error lógico. Si I_j está antes de I_i, T_j puede ejecutar con éxito su operación leer.
- I_j = escribir(Q). I_i e I_j están en conflicto. Si I_i está antes de I_j, T_j tendrá un error lógico. Si I_j está antes de I_i, T_j puede ejecutar con éxito su operación escribir.
- I_j = **borrar**(Q). I_i e I_j están en conflicto. Si I_i está antes de I_j, T_j tendrá un error lógico. Si I_j está antes de I_i, T_i tendrá un error lógico.
- I_j = **insertar**(Q). I_i e I_j están en conflicto. Supóngase que no existe el elemento de datos Q antes de la ejecución de I_i e I_j. Entonces si I_i está antes de I_j, hay un error lógico en T_i. Si I_j está antes de I_i, entonces no hay ningún error lógico. Igualmente si Q existía antes de la ejecución de I_i e I_j, entonces hay un error lógico si I_j está antes de I_i, pero no en otro caso.

Se puede concluir lo siguiente:

- En el protocolo de dos fases se necesita un bloqueo exclusivo en un elemento de datos antes de que se borre dicho elemento.
- En el protocolo de ordenación por marcas temporales se debe hacer una prueba similar que la que se hacía con escribir. Supóngase que la transacción T_i ejecuta **borrar**(Q).
  - — Si MT(T_i) < marca_temporal-L(Q), entonces el valor de Q que va a borrar T_i lo ha leído ya otra transacción T_j con MT(T_j) > MT(T_i). Por tanto se rechaza la operación **borrar** y T_i se retrocede.
  - — Si MT(T_i) < marca_temporal-E(Q), entonces la transacción T_j con MT(T_j) > MT(T_i) ha escrito Q. Por tanto se rechaza esta operación **borrar** y T_i se retrocede.
  - — En otro caso se ejecuta la operación **borrar**.

16.7.2. Inserción

Ya se ha visto que una operación **insertar**(Q) está en conflicto con una operación **borrar**(Q). De forma similar **insertar**(Q) está en conflicto con las operaciones leer(Q) y escribir(Q). No se pueden realizar operaciones leer o escribir sobre un elemento de datos hasta que este último exista.

Puesto que **insertar**(Q) asigna un valor al elemento de datos Q, se trata **insertar** de forma similar a escribir desde el punto de vista del control de concurrencia:

- En el protocolo de bloqueo de dos fases, si T_i realiza una operación **insertar**(Q), se da a T_i un blo-

queo exclusivo sobre el elemento de datos Q recientemente creado.

- En el protocolo de ordenación por marcas temporales, si T_i realiza una operación **insertar**(Q), se fijan los valores marca_temporal-L(Q) y marca_temporal-E(Q) a MT(T_i).

16.7.3. El fenómeno fantasma

Considérese una transacción T_{29} que ejecuta la siguiente pregunta SQL a la base de datos bancaria:

> **select sum**(*saldo*)
> **from** *cuenta*
> **where** *nombre-sucursal* = 'Pamplona'

La transacción T_{29} necesita acceder a todas las tuplas de la relación *cuenta* que pertenezcan a la sucursal Pamplona.

Sea T_{30} una transacción que ejecuta la siguiente inserción SQL:

> **insert into** *cuenta*
> **values** ('Pamplona', C-201, 900)

Sea P una planificación que involucra a T_{29} y T_{30}. Se espera que haya un conflicto potencial debido a las razones siguientes:

- Si T_{29} utiliza la tupla que ha insertado recientemente T_{30} al calcular **sum**(*saldo*), entonces T_{29} lee el valor que ha escrito T_{30}. Así, en una planificación secuencial equivalente a S, T_{30} debe ir antes de T_{29}.
- Si T_{29} no utiliza la tupla que ha insertado recientemente T_{30} al calcular **sum**(*saldo*), entonces en una planificación secuencial equivalente a S, T_{29} debe ir antes de T_{30}.

El segundo caso de los dos es curioso. T_{29} y T_{30} no acceden a ninguna tupla común, ¡y sin embargo están en conflicto! En efecto, T_{29} y T_{30} están en conflicto en una tupla fantasma. Si se realiza el control de concurrencia con granularidad de tupla, no se detecta dicho conflicto. Este problema recibe el nombre de **fenómeno fantasma**.

Para evitar el fenómeno fantasma se permite que T_{29} impida a otras transacciones crear nuevas tuplas en la relación *cuenta* con *nombre-sucursal* = 'Pamplona'.

Para encontrar todas las tuplas de *cuenta* con *nombre-sucursal* = 'Pamplona', T_{29} debe buscar o bien en toda la relación *cuenta*, o al menos en un índice de la relación. Hasta ahora se ha asumido implícitamente que los únicos elementos de datos a los que accede una transacción son tuplas. Sin embargo T_{29} es un ejemplo de transacción que lee información acerca de qué tuplas pertenecen a una relación, y T_{30} es un ejemplo de transacción que actualiza dicha información.

Claramente no es suficiente bloquear las tuplas a las que se accede; se necesita también bloquear la información acerca de qué tuplas pertenecen a la relación.

La solución más simple a este problema consiste en asociar un elemento de datos con la propia relación; el elemento de datos representa la información utilizada para encontrar las tuplas en la relación. Las transacciones como T_{29}, que lean la información acerca de qué tuplas pertenecen a la relación, tendrían que bloquear el elemento de datos correspondientes a la relación en modo compartido. Las transacciones como T_{30}, que actualicen la información acerca de qué tuplas pertenecen a la relación, tendrían que bloquear el elemento de datos en modo exclusivo. De este modo T_{29} y T_{30} tendrían un conflicto en un elemento de datos real, en lugar de tenerlo en uno fantasma.

No se debe confundir el bloqueo de una relación completa, como en el bloqueo de granularidad múltiple, con el bloqueo del elemento de datos correspondiente a la relación. Al bloquear el elemento de datos la transacción sólo evita que otras transacciones actualicen la información acerca de qué tuplas pertenecen a la relación. Sigue siendo necesario un bloqueo de tuplas. Una transacción que acceda directamente a una tupla puede obtener un bloqueo sobre las tuplas incluso si otra transacción posee un bloqueo exclusivo sobre el elemento de datos correspondiente a la propia relación.

El inconveniente principal de bloquear un elemento de datos correspondiente a la relación es el bajo grado de concurrencia —se impide que dos transacciones que inserten distintas tuplas en la relación se ejecuten concurrentemente.

Una solución mejor es la técnica de **bloqueo de índice**. Toda transacción que inserte una tupla en una relación debe insertar información en cada uno de los índices que se mantengan en la relación. Se elimina el fenómeno fantasma al imponer un protocolo para los índices. Para simplificar, sólo se van a considerar los índices del árbol B^+.

Como se vio en el Capítulo 12, todo valor de la clave de búsqueda se asocia a un nodo hoja índice. Una consulta usará normalmente uno o más índices para acceder a la relación. Una inserción debe insertar una nueva tupla en todos los índices de la relación. En el ejemplo se asume que hay un índice para *cuenta* en *nombre-sucursal*. Entonces T_{30} debe modificar la hoja que contiene la clave Pamplona. Si T_{29} lee el mismo nodo hoja para localizar todas las tuplas que pertenecen a la sucursal Pamplona, entonces T_{29} y T_{30} tienen un conflicto en dicho nodo hoja.

El **protocolo de bloqueo de índice** toma las ventajas de la disponibilidad de índices en una relación convirtiendo las apariciones del fenómeno fantasma en conflictos en los bloqueos sobre los nodos hoja índice. El protocolo opera de la siguiente manera:

- Toda relación debe tener al menos un índice.
- Una transacción T_i puede acceder a las tuplas de una relación únicamente después de haberlas encontrado primero a través de uno o más índices de la relación.
- Una transacción T_i que realiza una búsqueda (o bien una búsqueda de rango o una búsqueda concreta) debe bloquear en modo compartido todos los nodos hoja índice a los que accede.
- Una transacción T_i no puede insertar, borrar o actualizar una tupla t_i en una relación r sin actualizar todos los índices de r. La transacción debe obtener bloqueos en modo exclusivo sobre todos los nodos hoja índice que están afectados por la inserción, el borrado o la actualización. Para la inserción y el borrado, los nodos hoja afectados son aquellos que contienen (después de la inserción) o han contenido (antes de la modificación) el valor de la clave de búsqueda en la tupla. Para las actualizaciones, los nodos hoja afectados son los que (antes de la modificación) contienen el valor antiguo de la clave de búsqueda y los nodos que (después de la modificación) contienen el nuevo valor de la clave de búsqueda.
- Hay que cumplir las reglas del protocolo de bloqueo de dos fases.

Existen variantes de la técnica de bloqueo de índice para eliminar el fenómeno fantasma con otros protocolos de control de concurrencia que se han presentado en este capítulo.

16.8. NIVELES DÉBILES DE CONSISTENCIA

La secuencialidad es un concepto útil porque permite a los programadores ignorar los problemas relacionados con la concurrencia cuando codifican las transacciones. Si todas las transacciones tienen la propiedad de mantener la consistencia de la base de datos si se ejecutan por separado, la secuencialidad asegura que las ejecuciones concurrentes mantienen la consistencia. Sin embargo, puede que los protocolos necesarios para asegurar la secuencialidad permitan muy poca concurrencia para algunas aplicaciones. En estos casos se utilizan los niveles más débiles de consistencia. El uso de niveles más débiles de consistencia añade una nueva carga a los programadores para asegurar la corrección de las bases de datos.

16.8.1. Consistencia de grado dos

El objetivo de la **consistencia de grado dos** es evitar abortar en cascada sin asegurar necesariamente la secuencialidad. El protocolo de bloqueo para la consistencia de grado dos utiliza los mismos dos modos de bloqueo que se utilizan para el protocolo de bloqueo de dos fases: compartido (C) y exclusivo (X). Las transacciones deben mantener el modo de bloqueo adecuado cuando tengan acceso a un elemento de datos.

A diferencia de la situación en los bloqueos de dos fases, los bloqueos-C pueden liberarse en cualquier momento y también se pueden establecer bloqueos en cualquier momento. Los bloqueos exclusivos no se pueden liberar hasta que la transacción se comprometa o se aborte. La secuencialidad no queda asegurada por este protocolo. En realidad, una transacción puede leer dos veces el mismo elemento de datos y obtener resultados diferentes. En la Figura 16.20, T_3 lee el valor de Q antes y después de que T_4 escriba su valor.

La posibilidad de que se produzca inconsistencia con la consistencia de grado dos hace que este enfoque no sea conveniente para muchas aplicaciones.

16.8.2. Estabilidad del cursor

La **estabilidad del cursor** es una forma de consistencia de grado dos diseñada para programas escritos en lenguajes de propósito general, los cuales iteran sobre las tuplas de una relación utilizando cursores. En vez de bloquear toda la relación, la estabilidad del cursor asegura que

- La tupla que está procesando la iteración esté bloqueada en modo compartido.
- Todas las tuplas modificadas estén bloqueadas en modo exclusivo hasta que se comprometa la transacción.

16.8.3. Niveles débiles de consistencia en SQL

La norma SQL también permite que una transacción especifique si puede ser ejecutada de tal forma que se convierta en no secuenciable con respecto a otras transacciones. Por ejemplo, una transacción puede operar en el nivel **sin compromiso de lectura**, lo que permite que la transacción lea registros incluso si éstos no se

T_3	T_4
bloquear-C(Q) leer(Q) desbloquear(Q)	
	bloquear-X(Q) leer(Q) escribir(Q) desbloquear(Q)
bloquear-C(Q) leer(Q) desbloquear(Q)	

FIGURA 16.20. Planificación no secuenciable con consistencia de grado dos.

han comprometido. SQL proporciona tales características para transacciones largas cuyos resultados no necesitan ser precisos. Por ejemplo, una información aproximada suele ser suficiente para las estadísticas utilizadas en la optimización de consultas. Si estas transacciones se ejecutaran en modo secuencial, podrían interferir con otras transacciones, provocando que la ejecución de las otras se retrasara.

Los niveles de consistencia especificados por SQL-92 son los siguientes:

- **Secuenciable** es el predeterminado.
- **Lectura repetible** sólo permite leer registros comprometidos, y además requiere que, entre dos lecturas de un registro realizadas por una transacción, no se permita que ninguna otra transacción actualice el registro. Sin embargo, la transacción puede no ser secuenciable con respecto a otras transacciones. Por ejemplo, cuando está buscando registros que satisfagan algunas condiciones, una transacción podría encontrar algunos de los registros que ha insertado una transacción comprometida, pero podría no encontrar otros.
- **Compromiso de lectura** sólo permite leer registros comprometidos, pero ni siquiera requiere lecturas repetibles. Por ejemplo, entre dos lecturas de un registro realizadas por una transacción, los registros deben ser actualizados por otras transacciones comprometidas. Esto es básicamente lo mismo que la consistencia de grado dos; la mayoría de los sistemas que soportan este nivel de consistencia debería implementar en realidad estabilidad del cursor, que es un caso especial de consistencia de grado dos.
- **Sin compromiso de lectura** permite incluso leer registros no comprometidos. Éste es el nivel de consistencia más bajo que permite SQL-92.

16.9. CONCURRENCIA EN ESTRUCTURAS DE ÍNDICE**

Es posible tratar el acceso a las estructuras de índice como el de otras estructuras de base de datos y aplicar las técnicas de control de concurrencia que se han descrito anteriormente. Sin embargo, puesto que se accede frecuentemente a los índices, se pueden convertir en un punto con mucho bloqueo, lo que produce un bajo grado de concurrencia. Por suerte, no es necesario tratar a los índices como a las demás estructuras de base de datos. Es perfectamente aceptable que una transacción busque en un índice dos veces y se encuentre con que la estructura del índice ha cambiado entre ambas búsquedas, mientras la búsqueda devuelva el conjunto correcto de tuplas. De este modo se acepta tener un acceso no secuenciable a un índice mientras siga siendo correcto dicho índice.

A continuación se mostrarán dos técnicas para tratar los accesos concurrentes a árboles B^+. En las notas bibliográficas se hace referencia a otras técnicas para árboles B^+, así como a técnicas para otras estructuras de índice.

Las técnicas que se presentan para el control de concurrencia en los árboles B^+ se basan en el bloqueo, pero no se emplean ni el bloqueo de dos fases ni el protocolo de árbol. Los algoritmos de búsqueda, inserción y borrado son los mismos que se usan en el Capítulo 12 con sólo algunas pequeñas modificaciones.

La primera técnica se denomina **protocolo del cangrejo**:

- Cuando se busca un valor clave, el protocolo del cangrejo bloquea primero el nodo raíz en modo compartido. Cuando se recorre el árbol hacia abajo, adquiere un bloqueo compartido sobre el siguiente nodo hijo. Después de adquirir el bloqueo sobre el nodo hijo, libera el bloqueo sobre el nodo padre. Repite este proceso hasta que alcanza un nodo hoja.
- Cuando se inserta o se borra un valor clave, el protocolo del cangrejo realiza estas acciones:
  - — Sigue el mismo protocolo que para la búsqueda hasta que alcanza el nodo hoja deseado. Hasta este punto, obtiene (y libera) tan sólo bloqueos compartidos.
  - — Bloquea el nodo hoja en modo exclusivo e inserta o borra el valor clave.
  - — Si necesita dividir un nodo o combinarlo con sus hermanos, o redistribuir los valores claves entre hermanos, el protocolo del cangrejo bloquea al padre del nodo en modo exclusivo. Después de realizar estas acciones, libera los bloqueos sobre el nodo y los hermanos.

    Si el padre requiere división, combinación o redistribución de valores clave, el protocolo mantiene el bloqueo sobre el padre, y la división, la combinación o la redistribución se sigue propagando de la misma manera. En otro caso, libera el bloqueo sobre el padre.

El protocolo obtiene su nombre de la forma en que los cangrejos avanzan moviéndose de lado, moviendo las patas de un lado, después las patas del otro, y así alternando sucesivamente. El avance de los bloqueos mientras el protocolo baja por el árbol o sube de nuevo (en el caso de divisiones, combinaciones o redistribuciones) actúa de forma similar a la del cangrejo.

Una vez que una operación particular libera un bloqueo sobre un nodo, otras operaciones pueden acceder

a ese nodo. Existe una posibilidad de interbloqueos entre las operaciones de búsqueda que bajan por el árbol, y las divisiones, combinaciones y redistribuciones que se propagan hacia arriba por el árbol. El sistema puede manejar con facilidad tales interbloqueos reiniciando la operación de búsqueda desde la raíz, después de liberar los bloqueos mantenidos por la operación.

La segunda técnica consigue aún más concurrencia, impidiendo incluso que se mantenga un bloqueo sobre un nodo mientras se está adquiriendo el bloqueo sobre otro nodo, utilizando una versión modificada de los árboles B^+ llamados **árboles B enlazados**; los árboles B enlazados requieren que todo nodo (incluyendo los nodos internos, no sólo las hojas) mantenga un puntero a su hermano derecho. Se necesita este puntero porque una búsqueda que tenga lugar mientras se divide un nodo puede que tenga que buscar no sólo ese nodo sino también el hermano derecho de ese nodo (si existe alguno). Esta técnica se va a ilustrar con un ejemplo después de presentar los procedimientos modificados del **protocolo de bloqueo con árboles B enlazados**.

- **Búsqueda.** Se debe bloquear en modo compartido cada nodo del árbol B^+ antes de que se acceda a él. Dicho bloqueo se libera antes de que se solicite algún otro bloqueo sobre algún nodo del árbol B^+. Si tiene lugar una división de forma concurrente con una búsqueda, el valor de la clave de búsqueda deseado puede dejar de aparecer dentro del rango de valores que representa un nodo que se ha accedido en la búsqueda. En tal caso se representa el valor de la clave de búsqueda por medio de un nodo hermano, el cual coloca el sistema siguiendo el puntero al hermano derecho. Sin embargo, el sistema bloquea los nodos hoja siguiendo el protocolo de bloqueo de dos fases, como se describe en el Apartado 16.7.3, para evitar el fenómeno fantasma.
- **Inserción y borrado.** El sistema sigue las reglas de la búsqueda para localizar el nodo sobre el cual se va a realizar la inserción o el borrado. Se modifica el bloqueo en modo compartido sobre ese nodo a modo exclusivo y se realiza la inserción o el borrado. Se bloquean los nodos hoja afectados por la inserción o el borrado siguiendo el protocolo de bloqueo de dos fases, como se describe en el Apartado 16.7.3, para evitar el fenómeno fantasma.
- **División.** Si se divide un nodo se crea uno nuevo siguiendo el algoritmo del Apartado 12.3 y se convierte en el hermano derecho del nodo original. Se fijan los punteros al hermano derecho del nodo original (ya que es un nodo interno; los nodos hoja se bloquean en dos fases) y del nuevo nodo. Seguidamente se libera el bloqueo en modo exclusivo sobre el nodo original y se solicita un bloqueo sobre el padre para que se pueda insertar un nuevo nodo. (No hay necesidad de bloquear o desbloquear el nuevo nodo.)
- **Fusión.** Si un nodo tiene muy pocos valores de clave de búsqueda después de un borrado, se debe bloquear en modo exclusivo el nodo con el que se debe fusionar. Una vez que se fusionen estos nodos se solicita un bloqueo en modo exclusivo sobre el padre para que se pueda eliminar el nodo borrado. En ese momento se libera el bloqueo sobre el nodo fusionado. Se libera el bloqueo sobre el nodo padre a no ser que también se tenga que fusionar.

Es importante observar que una inserción o un borrado pueden bloquear un nodo, desbloquearlo y posteriormente volverlo a bloquear. Además una búsqueda que se ejecute concurrentemente con operaciones de división o de combinación puede observar que la clave de búsqueda deseada se ha trasladado al nodo hermano derecho debido a la división o a la combinación.

Como ejemplo considérese el árbol B^+ de la Figura 16.21. Supóngase que hay dos operaciones concurrentes sobre dicho árbol B^+:

1. Insertar «Cádiz»
2. Buscar «Daimiel»

Considérese que la operación inserción comienza en primer lugar. Realiza una búsqueda de «Cádiz» y encuentra que el nodo en el cual se debe insertar «Cádiz» está vacío. Por tanto convierte el bloqueo compartido sobre el nodo en un bloqueo exclusivo y crea un nuevo nodo. El nodo original contiene ahora los valores de clave de búsqueda «Barcelona» y «Cádiz». El nuevo nodo contiene el valor de clave de búsqueda «Daimiel».

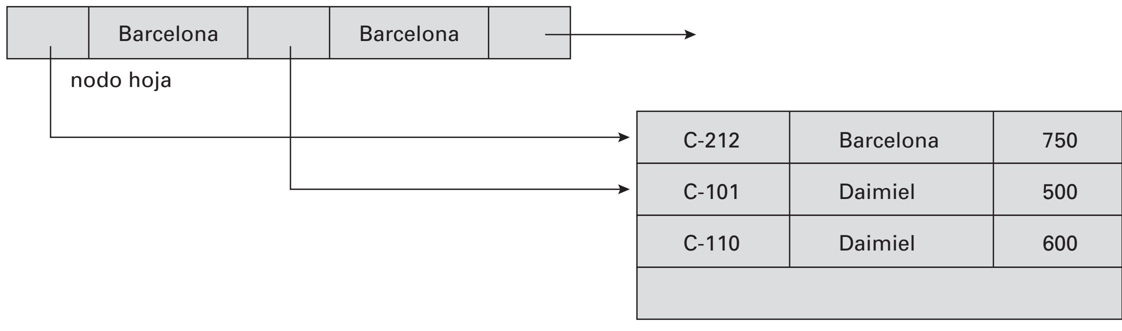


FIGURA 16.21. Nodo hoja para el índice del árbol B^+ de *cuenta* ($n = 3$).

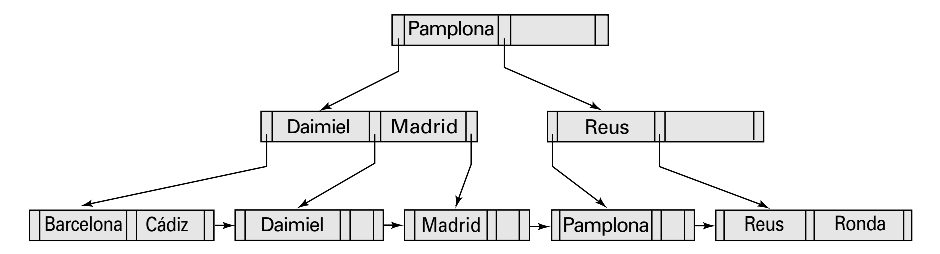


FIGURA 16.22. Inserción de «Cádiz» en el árbol B+ de la Figura 16.21.

Supóngase ahora que se produce un cambio de contexto que le pasa el control a la operación búsqueda. Dicha operación búsqueda accede a la raíz y sigue el puntero al hijo izquierdo de la raíz. Accede entonces a ese nodo y obtiene el puntero al hijo izquierdo. El hijo izquierdo contenía originalmente los valores de clave de búsqueda «Barcelona» y «Daimiel». Puesto que la operación inserción está bloqueando actualmente en modo exclusivo dicho nodo, la operación búsqueda debe esperar. Nótese que, en este punto, ¡la operación búsqueda no posee ningún bloqueo!

Ahora la operación inserción desbloquea el nodo hoja y vuelve a bloquear a su padre en esta ocasión en modo exclusivo. Se completa la inserción, lo que deja al árbol B+ como se muestra en la Figura 16.22. Continúa la operación de búsqueda. Sin embargo, tiene un puntero a un nodo hoja incorrecto. Sigue, por tanto, el puntero al hermano derecho para encontrar el nodo siguiente. Si este nodo es también incorrecto, se sigue también el puntero al hermano derecho. Se puede demostrar que, si una búsqueda tiene un puntero a un nodo incorrecto entonces, al seguir los punteros al hermano derecho, la búsqueda llega finalmente al nodo correcto.

Las operaciones de búsqueda y de inserción no pueden llevar a un interbloqueo. La combinación de nodos durante el borrado puede provocar inconsistencias, dado que una búsqueda puede tener que leer un puntero a un nodo borrado desde su padre, antes de que el nodo padre sea actualizado y, entonces, puede intentar acceder al nodo borrado. La búsqueda se tendría que reiniciar entonces desde la raíz. Dejar los nodos sin combinar evita tales inconsistencias. Esta solución genera nodos que contienen muy pocos valores clave de búsqueda y que violan algunas propiedades de los árboles B+. Sin embargo, en la mayoría de las bases de datos las inserciones son más frecuentes que los borrados, por lo que es probable que los nodos que tienen muy pocos valores claves de búsqueda ganen valores adicionales de forma relativamente rápida.

En lugar de bloquear nodos hoja índice en dos fases, algunos esquemas de control de concurrencia de índices utilizan bloqueo de valores clave sobre valores claves individuales, permitiendo que se inserten o se borren otros valores clave de la misma hoja. Por lo tanto, el bloqueo de valores clave proporciona una concurrencia mejorada. Sin embargo, utilizar el bloqueo de valores clave ingenuamente podría permitir que se produjera el fenómeno fantasma; para prevenir el fenómeno fantasma se utiliza la técnica bloqueo de la siguiente clave. En esta técnica, cada búsqueda por índice debe bloquear no sólo las claves encontradas dentro del rango (o la única clave, en caso de una búsqueda concreta), sino también el siguiente valor clave, esto es, el valor clave que es justo mayor que el último valor clave que estaba dentro del rango. Además, cada inserción no sólo debe bloquear el valor que se inserta, sino también el siguiente valor clave. Así, si una transacción intenta insertar un valor que estaba dentro del rango de la búsqueda por índice de otra transacción, las dos transacciones entrarán en conflicto en el valor clave que sigue al valor clave insertado. De forma similar, los borrados también deben bloquear el siguiente valor clave al valor que se ha borrado, para asegurar que se detecten los conflictos con las subsiguientes búsquedas de rango de otras consultas.

16.10. RESUMEN

- Cuando se ejecutan concurrentemente varias transacciones en la base de datos, puede dejar de conservarse la consistencia de los datos. Es necesario que el sistema controle la interacción entre las transacciones concurrentes, y dicho control se lleva a cabo mediante uno de los muchos mecanismos llamados esquemas de *control de concurrencia*.
- Se pueden usar varios esquemas de control de concurrencia para asegurar la secuencialidad. Todos estos esquemas o bien retrasan una operación o bien abortan la transacción que ha realizado la operación. Los más comunes son los protocolos de bloqueo, los esquemas de ordenación por marcas temporales, las técnicas de validación y los esquemas multiversión.
- Un protocolo de bloqueo es un conjunto de reglas, las cuales indican el momento en el que una transacción puede bloquear o desbloquear un elemento de datos de la base de datos.

- El protocolo de bloqueo de dos fases permite que una transacción bloquee un nuevo elemento de datos sólo si todavía no ha desbloqueado ningún otro elemento de datos. Este protocolo asegura la secuencialidad pero no la ausencia de interbloqueos. En ausencia de información acerca de la forma en que se accede a los elementos de datos, el protocolo de bloqueo de dos fases es necesario y suficiente para asegurar la secuencialidad.
- El protocolo de bloqueo estricto de dos fases permite liberar bloqueos exclusivos sólo al final de la transacción, para asegurar la recuperabilidad y la ausencia de cascadas en las planificaciones resultantes. El protocolo de bloqueo riguroso de dos fases libera todos los bloqueos sólo al final de la transacción.
- El esquema de ordenación por marcas temporales asegura la secuencialidad seleccionando previamente un orden entre todo par de transacciones. Se asocia una única marca temporal fija a cada transacción del sistema. Las marcas temporales de las transacciones determinan el orden de secuencialidad. De este modo, si la marca temporal de la transacción T_i es más pequeña que la de la transacción T_j, entonces el esquema asegura que la planificación que ha producido es equivalente a una planificación secuencial en la cual la transacción T_i aparece antes de la transacción T_j. Lo asegura retrocediendo una transacción siempre que se viole dicho orden.
- Un esquema de validación es un método de control de concurrencia adecuado en aquellos casos en los que la mayoría de las transacciones son de sólo lectura, y por tanto la tasa de conflictos entre dichas transacciones es baja. Se asocia una única marca temporal fija a cada transacción del sistema. Se determina el orden de secuencialidad por medio de la marca temporal. Nunca se retrasa una transacción en dicho esquema. Debe pasar, sin embargo, una comprobación de validación para poder completarse. Si no pasa la comprobación de validación se retrocede a su estado inicial.
- Hay circunstancias bajo las cuales puede ser conveniente agrupar varios elementos de datos y tratarlos como un conjunto de elementos de datos por motivos del trabajo, lo que da lugar a varios niveles de *granularidad*. Se permiten elementos de datos de varios tamaños y se define una jerarquía de elementos de datos en la cual los elementos más pequeños están anidados dentro de otros más grandes. Dicha jerarquía se puede representar de forma gráfica como un árbol. El orden de obtención de los bloqueos es desde la raíz hasta la hojas; se liberan desde las hojas hasta la raíz. Este protocolo asegura la secuencialidad pero no la ausencia de interbloqueos.
- Un esquema de control de concurrencia multiversión se basa en crear una nueva versión de un elemento de datos cada vez que una transacción va a escribir dicho elemento. Cuando se realiza una operación de lectura, el sistema elige una de las versiones para que se lea. El esquema de control de concurrencia asegura que la versión que se va a leer se elige de forma que asegure la secuencialidad usando las marcas temporales. Una operación de lectura tiene éxito siempre.
  - — En la ordenación por marcas temporales multiversión, una operación de escritura puede provocar el retroceso de una transacción.
  - — En el bloqueo de dos fases multiversión las operaciones de escritura pueden provocar una espera con bloqueo o posiblemente un interbloqueo.
- Algunos de los protocolos de bloqueo no evitan los interbloqueos. Una forma de prevenir los interbloqueos es utilizar una ordenación de los elementos de datos, y solicitar los bloqueos en una secuencia consistente con la ordenación.
- Otra forma de prevenir los interbloqueos es utilizar expropiación y retroceso de transacciones. Para controlar la expropiación se asigna una única marca temporal a cada transacción. El sistema utiliza estas marcas temporales para decidir si una transacción debe esperar o retroceder. Si una transacción se retrocede conserva su marca temporal *anterior* cuando vuelve a comenzar. El esquema herir-esperar es un esquema de expropiación.
- Si no se pueden prevenir los interbloqueos, el sistema debe ocuparse de ellos utilizando el esquema de detección y recuperación de interbloqueos. Para hacer esto, el sistema construye un grafo de espera. Un sistema está en estado de interbloqueo si y sólo si contiene un ciclo en el grafo de espera. Cuando el algoritmo de detección de interbloqueos determina que existe un interbloqueo, el sistema debe recuperarse del interbloqueo. Esto se lleva a cabo retrocediendo una o más transacciones para romper el interbloqueo.
- Se puede realizar una operación **borrar** sólo si la transacción que borra la tupla tiene un bloqueo en modo exclusivo sobre dicha tupla. A la transacción que inserta una nueva tupla se le concede un bloqueo en modo exclusivo sobre dicha tupla.
- Las inserciones pueden provocar el fenómeno fantasma, en el cual hay un conflicto entre una inserción y una pregunta incluso si las dos transacciones no acceden a tuplas comunes. Tales conflictos no se pueden detectar si el bloqueo se ha hecho sólo sobre tuplas a las que han accedido transacciones. Es necesario bloquear los datos utilizados para encontrar las tuplas en la relación. La técnica del bloqueo de índice resuelve este problema al exigir bloqueos sobre ciertos cajones de índices. Estos bloqueos aseguran que todas las transacciones conflictivas están en conflicto por un elemento de datos real en lugar de por uno fantasma.
- Los niveles débiles de consistencia se utilizan en algunas aplicaciones cuando la consistencia de los resultados de la consulta no es crítica, y utilizar secuen-

cialidad podría dar lugar a consultas que afectaran desfavorablemente al procesamiento de transacciones. La consistencia de grado dos es uno de los niveles de consistencia débiles; la estabilidad de cursor es un caso especial de consistencia de grado dos y se utiliza bastante. SQL:1999 permite a las consultas especificar el nivel de consistencia requerido.

• Se pueden desarrollar técnicas de control de concurrencia para estructuras especiales. A menudo se aplican técnicas especiales en los árboles B^+ para permitir una mayor concurrencia. Estas técnicas permiten accesos no secuenciables al árbol B^+, pero aseguran que la estructura del árbol B^+ es correcta y que los accesos a la base de datos son secuenciables.

TÉRMINOS DE REPASO

- Bloqueo
  - Compatibilidad
  - Concesión
  - Espera
  - Solicitud
- Bloqueo de dos fases multiversión
  - Transacciones de actualización
  - Transacciones de sólo lectura
- Concurrencia en índices
  - Árboles B enlazados
  - Bloqueo de la siguiente clave
  - Cangrejo
  - Protocolo de bloqueo con árboles B enlazados
- Control de concurrencia
- Control de concurrencia multiversión
- Conversión de bloqueo
  - Bajar
  - Subir
- Detección de interbloqueos
  - Grafo de espera
- Fenómeno fantasma
  - Protocolo de bloqueo de índice
  - Sin compromiso de lectura
- Granularidad múltiple
  - Bloqueos explícitos
  - Bloqueos implícitos
  - Bloqueos intencionales
- Inanición
- Interbloqueo
- Marca temporal
  - Contador lógico
  - marca_temporal-E(Q)
  - marca_temporal-L(Q)
  - Reloj del sistema
- Modos de bloqueo intencionales
  - Intencional-compartido (IC)
  - Intencional-exclusivo (IX)
  - Intencional-exclusivo y compartido (IXC)
- Niveles débiles de consistencia
  - Compromiso de lectura
  - Consistencia de grado dos
  - Estabilidad del cursor
  - Lectura repetible
- Operaciones para insertar y borrar
- Ordenación por marcas temporales multiversión
- Planificación legal
- Prevención de interbloqueos
  - Bloqueos ordenados
  - Esquema esperar-morir
  - Esquema herir-esperar
  - Esquemas basados en tiempo límite
  - Expropiación de bloqueos
- Protocolo de bloqueo
- Protocolo de bloqueo de dos fases
  - Bloqueo estricto de dos fases
  - Bloqueo riguroso de dos fases
  - Fase de crecimiento
  - Fase de decrecimiento
- Protocolo de bloqueo de granularidad múltiple
- Protocolo de ordenación por marcas temporales
  - Regla de escritura de Thomas
- Protocolos basados en grafos
  - Dependencia de compromiso
  - Protocolo de árbol
- Protocolos basados en validación
  - Comprobación de validación
  - Fase de escritura
  - Fase de lectura
  - Fase de validación
- Protocolos basados en marcas temporales
- Punto de bloqueo
- Tipos de bloqueo
  - Bloqueo en modo compartido (C)
  - Bloqueo en modo exclusivo (X)
- Recuperación de interbloqueos
  - Retroceso parcial
  - Retroceso total
- Tratamiento de interbloqueos
  - Detección
  - Prevención
  - Recuperación
- Versiones

EJERCICIOS

16.1. Demuéstrese que el protocolo de bloqueo de dos fases asegura la secuencialidad en cuanto a conflictos y que se pueden secuenciar las transacciones a través de sus puntos de bloqueo.

16.2. Considérense las dos transacciones siguientes:

T_{31}: leer(A);
leer(B);
si A = 0
entonces B := B + 1;
escribir(B)

T_{32}: leer(B);
leer(A);
si B = 0
entonces A := A + 1;
escribir(A).

Añádanse a las transacciones T_{31} y T_{32} las instrucciones de bloqueo y desbloqueo para que sigan el protocolo de dos fases. ¿Puede producir la ejecución de estas transacciones un interbloqueo?

16.3. ¿Qué beneficio proporciona el bloqueo estricto de dos fases? ¿Qué inconvenientes tiene?

16.4. ¿Qué beneficio proporciona el bloqueo riguroso de dos fases? Compárese con otras formas de bloqueo de dos fases.

16.5. Muchas implementaciones de sistemas de bases de datos utilizan el bloqueo estricto de dos fases. Indíquense tres razones que expliquen la popularidad de este protocolo.

16.6. Considérese una base de datos organizada como un árbol con raíz. Supóngase que se inserta un nodo ficticio entre cada par de nodos. Demuéstrese que, si se sigue el protocolo de árbol con este nuevo árbol, se obtiene mayor concurrencia que con el árbol original.

16.7. Demuéstrese con un ejemplo que hay planificaciones que son posibles con el protocolo de árbol que no lo son con otros protocolos de bloqueo de dos fases y viceversa.

16.8. Considérese la siguiente extensión del protocolo de bloqueo de árbol que permite bloqueos compartidos y exclusivos:

- Una transacción puede ser de sólo lectura, en cuyo caso sólo puede solicitar bloqueos compartidos, o bien puede ser de actualización, en cuyo caso sólo puede solicitar bloqueos exclusivos.
- Cada transacción debe seguir las reglas del protocolo de árbol. Las transacciones de sólo lectura deben bloquear primero cualquier elemento de datos, mientras que las transacciones de actualización deben bloquear primero la raíz.

Demuéstrese que este protocolo asegura la secuencialidad y la ausencia de interbloqueos.

16.9. Considérese el siguiente protocolo de bloqueo basado en grafo, el cual sólo permite bloqueos exclusivos y que funciona con grafos de datos con forma de grafo dirigido acíclico con raíz.

- Una transacción puede bloquear en primer lugar cualquier nodo.
- Para bloquear cualquier otro nodo, la transacción debe poseer un bloqueo sobre la mayoría de los padres de dicho nodo.

Demuéstrese que este protocolo asegura la secuencialidad y la ausencia de interbloqueos.

16.10. Considérese el siguiente protocolo de bloqueo basado en grafos que sólo permite bloqueos exclusivos y que funciona con grafos de datos con forma de grafo dirigido acíclico con raíz.

- Una transacción puede bloquear en primer lugar cualquier nodo.
- Para bloquear cualquier otro nodo, la transacción debe haber visitado a todos los padres de dicho nodo y debe poseer un bloqueo sobre uno de los padres del vértice.

Demuéstrese que este protocolo asegura la secuencialidad y la ausencia de interbloqueos.

16.11. Considérese una variante del protocolo de árbol llamada protocolo de *bosque*. La base de datos está organizada como un bosque de árboles con raíz. Cada transacción T_i debe seguir las reglas siguientes:

- El primer bloqueo en un árbol puede hacerse sobre cualquier elemento de datos.
- Se pueden solicitar el segundo y posteriores bloqueos sólo si el padre del nodo solicitado está bloqueado actualmente.
- Se pueden desbloquear los elementos de datos en cualquier momento.
- T_i no puede volver a bloquear un elemento de datos después de haberlo desbloqueado.

Demuéstrese que el protocolo de bosque no asegura la secuencialidad.

16.12. El bloqueo no se hace explícitamente en lenguajes de programación persistentes. En vez de esto se deben bloquear los objetos (o sus páginas correspondientes) cuando se accede a dichos objetos. Muchos de los sistemas operativos más modernos permiten al usuario definir protecciones de acceso (sin acceso, lectura, escritura) para las páginas, y aquellos accesos a memoria que violen las protecciones de acceso dan como resultado una violación de protección (véase la orden `mprotect` de Unix, por ejemplo). Descríbase la forma en que se puede usar el mecanismo de protección de acceso para bloqueos a nivel de página en lenguajes de programación persistentes. (*Sugerencia*: La técnica es algo parecida a la que se usaba para el rescate hardware del Apartado 11.9.4.)

	C	X	I
C	cierto	falso	falso
X	falso	falso	falso
I	falso	falso	cierto

FIGURA 16.23. Matriz de compatibilidad de bloqueos.

16.13. Considérese una base de datos que tiene la operación atómica **incrementar** además de las operaciones leer y escribir. Sea V el valor del elemento de datos X. La operación

incrementar(X) en C

asigna el valor $V + C$ a X en un paso atómico. El valor de X no está disponible hasta que no se ejecute posteriormente una operación leer(X). En la Figura 16.17 se muestra una matriz de compatibilidad de bloqueos para tres tipos de bloqueo: modo compartido, exclusivo y de incremento.

a. Demuéstrese que, si todas las transacciones bloquean el dato al que acceden en el modo correspondiente, entonces el bloqueo de dos fases asegura la secuencialidad.

b. Demuéstrese que la inclusión del bloqueo en modo **incrementar** permite una mayor concurrencia. (Sugerencia: Considérense las transacciones de transferencia de fondos del ejemplo bancario.)

16.14. En la ordenación por marcas temporales, **marca_temporal-E**(Q) indica la mayor marca temporal de todas las transacciones que hayan ejecutado escribir(Q) con éxito. Supóngase que en lugar de ello, **marca_temporal-E**(Q) se define como la marca temporal de la transacción más reciente que haya ejecutado escribir(Q) con éxito. ¿Hay alguna diferencia al cambiar esta definición? Razónese la respuesta.

16.15. Cuando se retrocede una transacción en el protocolo de ordenación por marcas temporales se le asigna una nueva marca temporal. ¿Por qué no puede conservar simplemente su antigua marca temporal?

16.16. En el protocolo de granularidad múltiple, ¿qué diferencia hay entre bloqueo implícito y explícito?

16.17. Aunque el modo IXC es útil para el bloqueo de granularidad múltiple, no se usa un modo exclusivo e intencional-compartido (ICX). ¿Por qué no es útil?

16.18. La utilización de un bloqueo de granularidad múltiple puede necesitar más o menos bloqueos que en un sistema equivalente con una granularidad simple de bloqueo. Proporciónense ejemplos de ambas situaciones y compárese el aumento relativo de la concurrencia que se permite.

16.19. Considérese el esquema de control de concurrencia basado en la validación del Apartado 16.3. Demuéstrese que si se elige Validación(T_i) en lugar de Inicio(T_i) como marca temporal de la transacción T_i, se puede esperar una mejor respuesta en tiempo debido a que la tasa de conflictos entre las transacciones es realmente baja.

16.20. Demuéstrese que hay planificaciones que son posibles con el protocolo de bloqueo de dos fases que no lo son con el protocolo de marcas temporales y viceversa.

16.21. Para cada uno de los protocolos siguientes, descríbanse los aspectos de aplicación práctica que sugieran utilizar el protocolo y aspectos que sugieran no usarlo:

- Bloqueo de dos fases
- Bloqueo de dos fases con granularidad múltiple
- Protocolo de árbol
- Ordenación por marcas temporales
- Validación
- Ordenación por marcas temporales multiversión
- Bloqueo de dos fases multiversión

16.22. En una versión modificada del protocolo de marcas temporales se necesita comprobar un bit de compromiso para saber si una petición de **lectura** debe esperar o no. Explíquese cómo puede evitar el bit de compromiso que aborten en cascada. ¿Por qué no se necesita hacer esta comprobación con las peticiones de escritura?

16.23. Explíquese por qué la siguiente técnica de ejecución de transacciones puede proporcionar mayor rendimiento que la utilización del bloqueo estricto de dos fases: primero se ejecuta la transacción sin adquirir ningún bloqueo y sin realizar ninguna escritura en la base de datos como en las técnicas basadas en validación, pero a diferencia de las técnicas de validación no se realiza otra validación o escritura en la base de datos. En cambio, se vuelve a ejecutar la transacción utilizando bloqueo estricto de dos fases. (Sugerencia: Considérense esperas para la E/S de disco.)

16.24. ¿Bajo qué condiciones es menos costoso evitar los interbloqueos que permitirlos y luego detectarlos?

16.25. Si se evitan los interbloqueos, ¿sigue siendo posible que haya inanición? Razónese la respuesta.

16.26. Considérese el protocolo de ordenación por marcas temporales, y dos transacciones, una que escribe dos elementos de datos p y q, y otra que lee los mismos dos elementos de datos. Obténgase una planificación por medio de la cual la comprobación por marcas temporales para una operación **escribir** falle y provoque el reinicio de la primera transacción, provocando a su vez una cancelación en cascada de la otra transacción. Muéstrese cómo esto podría acabar en inanición de las dos transacciones. (Tal situación, donde dos o más procesos realizan acciones, pero no se puede completar la tarea porque se interacciona con otros procesos, se denomina **interbloqueo**.)

16.27. Explíquese el fenómeno fantasma. ¿Por qué produce este fenómeno una ejecución concurrente incorrecta a pesar de utilizar el protocolo de bloqueo de dos fases?

16.28. Diséñese un protocolo basado en marcas temporales que evite el fenómeno fantasma.

16.29. Explíquese la razón por la cual se utiliza la consistencia de grado dos. ¿Qué desventajas tiene esta técnica?

16.30. Supóngase que se utiliza el protocolo de árbol del Apartado 16.1.5 para administrar el acceso concurrente a un árbol B^+. Puesto que puede haber una división en

una inserción que afecte a la raíz, se deduce que una operación inserción no puede liberar ningún bloqueo hasta que se complete la operación entera. ¿Bajo qué circunstancias es posible liberar antes un bloqueo?

16.31. Dense ejemplos de planificaciones para mostrar que si cualquier búsqueda, inserción o borrado no bloquea el siguiente valor clave, el fenómeno fantasma podría ser indetectable.

NOTAS BIBLIOGRÁFICAS

Gray y Reuter [1993] proporcionan un libro de texto detallado que cubre conceptos de procesamiento de transacciones, incluyendo conceptos de control de concurrencia y detalles de implementación. Bernstein y Newcomer [1997] proporcionan un libro de texto que trata varios aspectos del procesamiento de transacciones incluyendo el control de concurrencia.

Entre los primeros libros de texto que incluyen discusiones sobre el control de concurrencia y la recuperación se incluyen los de Papadimitriou [1986] y Bernstein et al. [1987]. Gray [1978] presenta uno de los primeros estudios sobre aspectos de la implementación del control de concurrencia y la recuperación.

Eswaran et al. [1976] introdujo el protocolo de bloqueo de dos fases. El protocolo de bloqueo de árbol es de Silberschatz y Kedem [1980]. Yannakakis et al. [1979], Kedem y Silberschatz [1983] y Buckley y Silberschatz [1985] desarrollaron otros protocolos de bloqueo que no son de dos fases y que operan con grafos más generales. Lien y Weinberger [1978], Yannakakis et al. [1979], Yannakakis [1981] y Papadimitriou [1982] ofrecen discusiones generales acerca de los protocolos de bloqueo. Korth [1983] explora varios modos de bloqueo que se pueden obtener a partir de los modos básicos, compartido y exclusivo.

El Ejercicio 16.6 es de Buckley y Silberschatz [1984]. El Ejercicio 16.8 es de Kedem y Silberschatz [1983]. El Ejercicio 16.9 es de Kedem y Silberschatz [1979]. El Ejercicio 16.10 es de Yannakakis et al. [1979]. El Ejercicio 16.13 es de Korth [1983].

El esquema de control de concurrencia basado en marcas temporales es de Reed [1983]. Bernstein y Goodman [1980] presentan una exposición de varios algoritmos de control de concurrencia basados en marcas temporales. Buckley y Silberschatz [1983] presentan un algoritmo de marcas temporales que no necesita retroceso para asegurar la secuencialidad. El esquema de control de concurrencia basado en validación es de Kung y Robinson [1981].

El protocolo de bloqueo para elementos de datos de granularidad múltiple es de Gray et al. [1975]. Gray et al. [1976] presentan una descripción detallada. Los efectos de la granularidad de bloqueo se discuten en Ries y Stonebraker [1976]. Korth [1983] formaliza el bloqueo con granularidad múltiple para una colección arbitraria de modos de bloqueo (que permite más funcionalidades además de simplemente leer y escribir). Esta aproximación incluye una clase de modos de bloqueo llamados modos de *actualización* para permitir conversión de bloqueos. Carey [1983] extiende la idea de granularidad múltiple a la de control de concurrencia basado en marcas temporales. Korth [1982] presenta una extensión del protocolo que asegura la ausencia de interbloqueos. Lee y Liou [1996] discuten el bloqueo de granularidad múltiple para sistemas de bases de datos orientados a objetos.

Bernstein et al. [1983] ofrecen discusiones acerca del control de concurrencia multiversión. En Silberschatz [1982] aparece un algoritmo de bloqueo de árbol multiversión. Reed [1978] y Reed [1983] introdujeron la ordenación por marcas temporales multiversión. Lai y Wilkinson [1984] describen un certificador de bloqueo de dos fases multiversión.

Dijkstra [1965] fue uno de los pioneros y de mayor influencia en el área de los interbloqueos. Holt [1971] y Holt [1972] fueron los primeros que formalizaron la idea de interbloqueo a través de un modelo de grafo similar al que se presenta en este capítulo. El algoritmo de detección de interbloqueos con marcas temporales es de Rosenkrantz et al. [1978]. Gray et al. [1981b] presenta un análisis de la probabilidad de tener esperas e interbloqueos. Fussell et al. [1981] y Yannakakis [1981] presentan resultados teóricos acerca de los interbloqueos y la secuencialidad. En libros de texto de algoritmos estándar se pueden encontrar algoritmos de detección de ciclos, como por ejemplo en Cormen et al. [1990].

En Gray et. al. [1975] se introduce la consistencia de grado dos. Los niveles de consistencia —o aislamiento— que ofrece SQL se explican y comentan en Berenson et al. [1995].

Bayer y Schkolnick [1977] y Johnson y Shasha [1993] estudiaron la concurrencia en árboles B^+. La técnica que se ha presentado en el Apartado 16.9 está basada en Kung y Lehman [1980], y en Lehman y Yao [1981]. En Mohan [1990a] y Mohan y Levine [1992] se describe la técnica del bloqueo del valor clave utilizada en ARIES que proporciona una gran concurrencia en el acceso a los árboles B^+.

Shasha y Goodman [1988] presentan una buena representación de protocolos de concurrencia para estructuras de índice. Ellis [1987] presenta una técnica de control de concurrencia para asociación lineal. Los árboles B enlazados se discuten en Lomet y Salzberg [1992]. En Ellis [1980a] y Ellis [1980b] aparecen algoritmos de control de concurrencia para otras estructuras de índice.

CAPÍTULO

17 SISTEMA DE RECUPERACIÓN

Una computadora, al igual que cualquier otro dispositivo eléctrico o mecánico, está sujeta a fallos. Éstos se producen por diferentes motivos como: fallos de disco, cortes de corriente, errores en el software, un incendio en la habitación de la computadora o incluso sabotaje. En cada uno de estos casos puede perderse información. Por tanto, el sistema de bases de datos debe realizar con anticipación acciones que garanticen que las propiedades de atomicidad y durabilidad de las transacciones, presentadas en el Capítulo 15, se preservan a pesar de tales fallos. Una parte integral de un sistema de bases de datos es un **esquema de recuperación**, el cual es responsable de la restauración de la base de datos al estado consistente previo al fallo. El esquema de recuperación también debe proporcionar **alta disponibilidad**; esto es, debe minimizar el tiempo durante el que la base de datos no se puede usar después de un fallo.

17.1. CLASIFICACIÓN DE LOS FALLOS

En un sistema pueden producirse varios tipos de fallos, cada uno de los cuales requiere un tratamiento diferente. El tipo de fallo más fácil de tratar es el que no conduce a una pérdida de información en el sistema. Los fallos más difíciles de tratar son aquellos que provocan una pérdida de información. En este capítulo consideraremos sólo los siguientes tipos de fallos:

- **Fallo en la transacción.** Hay dos tipos de errores que pueden hacer que una transacción falle:
  - **Error lógico.** La transacción no puede continuar con su ejecución normal a causa de alguna condición interna, como una entrada incorrecta, datos no encontrados, desbordamiento o exceso del límite de recursos.
  - **Error del sistema.** El sistema se encuentra en un estado no deseado (por ejemplo, de interbloqueo) como consecuencia del cual una transacción no puede continuar con su ejecución normal. La transacción, sin embargo, se puede volver a ejecutar más tarde.
- **Caída del sistema.** Un mal funcionamiento del hardware o un error en el software de la base de datos o del sistema operativo causa la pérdida del contenido de la memoria volátil y aborta el procesamiento de una transacción. El contenido de la memoria no volátil permanece intacto y no se corrompe.

  La suposición de que los errores de hardware o software fuercen una parada del sistema, pero no corrompan el contenido de la memoria no volátil, se conoce como **supuesto de fallo-parada**. Los sistemas bien diseñados tienen numerosas comprobaciones internas, al nivel de hardware y de software, que abortan el sistema cuando existe un error. De aquí que el supuesto de fallo-parada sea razonable.
- **Fallo de disco.** Un bloque del disco pierde su contenido como resultado de bien una colisión de la cabeza lectora, bien un fallo durante una operación de transferencia de datos. Las copias de los datos que se encuentran en otros discos o en archivos de seguridad en medios de almacenamiento secundarios, como cintas, se utilizan para recuperarse del fallo.

Para determinar cómo el sistema debe recuperarse de los fallos, es necesario identificar los modos de fallo de los dispositivos de almacenamiento. A continuación se verá cómo afectan estos modos de fallo a los contenidos de la base de datos. Entonces se pueden proponer algoritmos para garantizar la consistencia de la base de datos y la atomicidad de las transacciones a pesar de los fallos. Estos algoritmos se conocen como algoritmos de recuperación, aunque constan de dos partes:

1. Acciones llevadas a cabo durante el procesamiento normal de transacciones para asegurar que existe información suficiente para permitir la recuperación frente a fallos.
2. Acciones llevadas a cabo después de ocurrir un fallo para restablecer el contenido de la base de datos a un estado que asegure la consistencia de la base de datos, la atomicidad de la transacción y la durabilidad.

17.2. ESTRUCTURA DEL ALMACENAMIENTO

Como vimos en el Capítulo 10, los diferentes elementos que componen una base de datos pueden ser almacenados y accedidos con diferentes medios de almacenamiento. Para entender cómo se pueden garantizar las propiedades de atomicidad y durabilidad de una transacción, se deben comprender mejor estos medios de almacenamiento y sus métodos de acceso.

17.2.1. Tipos de almacenamiento

En el Capítulo 11 se vio que los medios de almacenamiento se pueden distinguir según su velocidad relativa, capacidad, y resistencia a fallos, y se pueden clasificar como almacenamiento volátil o no volátil. Se repasarán estos términos y se introducirá otra clase de almacenamiento, denominada almacenamiento estable.

- **Almacenamiento volátil.** La información que reside en almacenamiento volátil no suele sobrevivir a las caídas del sistema. La memoria principal y la memoria caché son ejemplos de este almacenamiento. El acceso al almacenamiento volátil es muy rápido, tanto por la propia velocidad de acceso a la memoria, como porque es posible acceder directamente a cualquier elemento de datos.
- **Almacenamiento no volátil.** La información que reside en almacenamiento no volátil sobrevive a las caídas del sistema. Los discos y las cintas magnéticas son ejemplos de este almacenamiento. Los discos se utilizan para almacenamiento en conexión, mientras que las cintas se usan para almacenamiento permanente. Ambos, sin embargo, pueden fallar (por ejemplo, colisión de la cabeza lectora), lo que puede conducir a una pérdida de información. En el estado actual de la tecnología, el almacenamiento no volátil es más lento en varios órdenes de magnitud que el almacenamiento volátil. Esta diferencia de velocidad es consecuencia de que los dispositivos de disco y de cinta sean electromecánicos, mientras que el almacenamiento volátil se basa por completo en circuitos integrados, como el almacenamiento volátil. En los sistemas de bases de datos los discos se utilizan fundamentalmente para el almacenamiento no volátil. Otros medios de almacenamiento no volátil sólo se usan normalmente para copias de seguridad de los datos. El almacenamiento *flash* (véase el Apartado 11.1), aunque no volátil, tiene capacidad insuficiente para la mayoría de los sistemas de bases de datos.
- **Almacenamiento estable.** La información que reside en almacenamiento estable *nunca* se pierde (bueno, *nunca diga nunca jamás*, porque teóricamente el *nunca* no puede garantizarse; por ejemplo, es posible, aunque extremadamente improbable, que un agujero negro se trague a la Tierra y ¡destruya para siempre todos los datos!). A pesar de que el almacenamiento estable es teóricamente imposible de conseguir, puede obtenerse una buena aproximación usando técnicas que hagan que la pérdida de información sea una posibilidad muy remota. La implementación del almacenamiento estable se discute en el Apartado 17.2.2.

Las diferencias entre los distintos tipos de almacenamiento son, con frecuencia, menos claras en la práctica que en la presentación anterior. Ciertos sistemas están provistos de una fuente de alimentación de seguridad, por lo que determinada memoria principal puede sobrevivir a las caídas del sistema y a cortes de corriente. Otras formas alternativas de almacenamiento no volátil, como los medios ópticos, ofrecen un grado de confianza incluso más alto que el de los discos.

17.2.2. Implementación del almacenamiento estable

Para implementar almacenamiento estable se debe replicar la información necesaria en varios medios de almacenamiento no volátil (normalmente discos) con modos de fallo independientes, y actualizar esa información de manera controlada para asegurar que un fallo durante una transferencia de datos no dañará la información necesaria.

Recuérdese (en el Capítulo 10) que los sistemas RAID (disposición redundante de discos independientes) garantizan que el fallo de un solo disco (incluso durante una transferencia de datos) no conduce a la pérdida de los datos. La variante más sencilla y rápida de un RAID es el disco con imagen, que guarda dos copias de cada bloque en distintos discos. Otras formas de RAID ofrecen menores costes a expensas de un rendimiento inferior.

Los sistemas RAID, sin embargo, no pueden proteger contra las pérdidas de datos debidas a desastres tales como un incendio o una inundación. Muchos sistemas de almacenamiento guardan copias de seguridad de las cintas en otro lugar como protección frente a tales desastres. No obstante, como las cintas no pueden ser trasladadas a otro lugar continuamente, los cambios que se hayan realizado después del último traslado de las cintas se perderán en caso de un desastre tal. Los sistemas más seguros guardan una copia de cada bloque de almacenamiento estable en un lugar remoto, escribiéndola a través de una red de computadoras, además de almacenar el bloque en un sistema de discos locales. Como los bloques se envían al sistema remoto al mismo tiempo y de la misma forma que se guardan en almacenamiento local, una vez que una operación de este tipo se com-

pleta los bloques copiados no pueden perderse, incluso en el caso de que ocurriese un desastre como un incendio o una inundación. En el Apartado 17.10 se estudian estos sistemas de *copia de seguridad remota*.

En el resto de este apartado se discute la manera de proteger a los medios de almacenamiento de los errores durante una transferencia de datos. Una transferencia de bloques entre la memoria y el disco puede acabar de diferentes formas:

- **Éxito.** La información transferida llega a su destino con seguridad.
- **Fallo parcial.** Ocurre un fallo en medio de la transferencia y el bloque de destino contiene información incorrecta.
- **Fallo total.** El fallo ocurre suficientemente pronto durante la transferencia para que el bloque de destino permanezca intacto.

Es necesario que, si se produce un **fallo durante una transferencia de datos**, el sistema lo detecte e invoque a un procedimiento de recuperación para restaurar el bloque a un estado estable. Para hacer esto, el sistema debe mantener dos bloques físicos por cada bloque lógico de la base de datos; en el caso de los discos con imagen, ambos bloques están en el mismo lugar; en el caso de copia de seguridad remota, uno de los bloques es local mientras que el otro está en un lugar remoto. Una operación de salida se ejecuta de la siguiente manera:

1. Se escribe la información en el primer bloque físico.
2. Cuando la primera escritura se completa con éxito, se escribe la misma información en el segundo bloque físico.
3. La salida está completada sólo después de que la segunda escritura finalice con éxito.

Durante la recuperación se examina cada par de bloques físicos. Si ambos coinciden y no existe ningún error detectable, entonces no son necesarias más acciones. Si un bloque contiene un error detectable, se reemplaza su contenido por el del segundo bloque. Este procedimiento de recuperación garantiza que la escritura en almacenamiento estable o bien se completa con éxito (esto es, se actualizan todas las copias) o bien no produce ningún cambio.

El requisito de comparar cada par correspondiente de bloques durante la recuperación es bastante costoso. Puede reducirse considerablemente ese coste si se registran las escrituras de bloques que están en progreso utilizando una pequeña cantidad de RAM no volátil. En la recuperación solamente es necesario comparar aquellos bloques para los que la escritura estuviera en progreso.

Los protocolos para escribir un bloque en un lugar remoto son similares a los utilizados para escribir bloques en un sistema de disco con imagen, que fueron examinados en el Capítulo 11 y, en particular, en el Ejercicio 11.4.

Este procedimiento puede extenderse fácilmente para permitir el uso de un número arbitrariamente alto de copias de cada bloque de almacenamiento estable. Aunque un elevado número de copias reduce la probabilidad de fallo incluso por debajo de la conseguida con dos copias, habitualmente es razonable la simulación de almacenamiento estable con sólo dos copias.

17.2.3. Acceso a los datos

Como se vio en el Capítulo 11, el sistema de bases de datos reside permanentemente en almacenamiento no volátil (normalmente discos) y se divide en unidades de almacenamiento de longitud fija denominadas **bloques**. Los bloques son las unidades de datos que se transfieren desde y hacia el disco y pueden contener varios elementos de datos. Supondremos que ningún elemento de datos mide dos o más bloques. Esta suposición es realista para la mayoría de las aplicaciones de procesamiento de datos tales como el ejemplo bancario.

Las transacciones llevan información del disco hacia la memoria principal y luego devuelven la información al disco. Las operaciones de entrada y salida se realizan en unidades de bloque. Nos referiremos a los bloques que residen en el disco como **bloques físicos**, y a los que residen temporalmente en la memoria principal como **bloques de memoria intermedia**. El área de memoria en donde los bloques residen temporalmente se denomina **memoria intermedia de disco**.

Las transferencias de un bloque entre disco y memoria principal se comienzan a través de las dos operaciones siguientes:

1. entrada(*B*) transfiere el bloque físico *B* a la memoria principal
2. salida(*B*) transfiere el bloque de memoria intermedia *B* al disco y reemplaza allí al correspondiente bloque físico.

Este esquema se ilustra en la Figura 17.1.

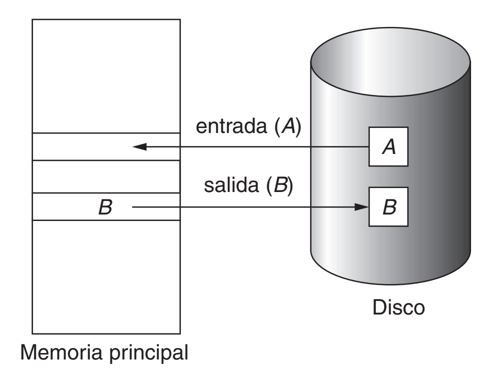


FIGURA 17.1. Operaciones de almacenamiento de bloques.

Cada transacción T_i posee un área de trabajo privada en la cual se guardan copias de todos los elementos de datos accedidos y actualizados por T_i. Esta área de trabajo se crea cuando se comienza una transacción y se elimina cuando la transacción o bien se compromete o bien aborta. Cada elemento de datos X almacenado en el área de trabajo de la transacción T_i se denotará como x_i. La transacción T_i interactúa con el sistema de bases de datos por medio de transferencias de datos desde su área de trabajo hacia la memoria intermedia del sistema y viceversa. Nosotros realizaremos transferencias de datos utilizando las dos operaciones siguientes:

1. leer(X) asigna el valor del elemento de datos X a la variable local x_i. Esta operación se ejecuta como sigue:
   a. Si el bloque B_X en el que reside X no está en la memoria principal, entonces se emite entrada(B_X).
   b. Asignar a x_i el valor de X en el bloque de memoria intermedia.
2. escribir(X) asigna el valor de la variable local x_i al elemento de datos X en el bloque de memoria intermedia. Esta operación se ejecuta como sigue:
   a. Si el bloque B_X en el que reside X no está en la memoria principal, entonces se lanza entrada(B_X).
   b. Asignar el valor de x_i a X en la memoria intermedia B_X.

Nótese que ambas operaciones pueden requerir la transferencia de un bloque desde disco a la memoria principal. En cambio, ninguna de ellas requiere específicamente la transferencia de un bloque desde la memoria principal al disco.

Un bloque de memoria intermedia B se escribe finalmente en el disco bien porque el gestor de la memoria intermedia necesita espacio en memoria para otros propósitos, o bien porque el sistema de base de datos desea reflejar en el disco los cambios sufridos por B. Diremos que el sistema de bases de datos **fuerza la salida** de la memoria intermedia B si ejecuta la orden salida(B).

Cuando una transacción necesita acceder a un elemento de datos X por primera vez, debe ejecutar leer(X). Todas las actualizaciones de X se llevan a cabo sobre x_i. Después de que la transacción acceda a X por última vez, se debe ejecutar escribir(X) para reflejar en la propia base de datos los cambios sufridos por X.

La operación salida(B_X) sobre el bloque de memoria intermedia B_X en el que reside X no tiene por qué tener efecto inmediatamente después de ejecutar escribir(X), ya que el bloque B_X puede contener otros elementos de datos que estén siendo accedidos todavía. Así, la salida real tiene lugar más tarde. Nótese que, si el sistema se bloquea después de ejecutar la operación escribir(X), pero antes de ejecutar salida(B_X), el nuevo valor de X nunca se escribe en el disco y, por tanto, se pierde.

17.3. RECUPERACIÓN Y ATOMICIDAD

Considérese de nuevo el sistema bancario simplificado y una transacción T_i que transfiere 50 € desde la cuenta A a la cuenta B, siendo los saldos iniciales de A y de B de 1.000 € y 2.000 € respectivamente. Supóngase que el sistema cae durante la ejecución de T_i después de haberse ejecutado salida(B_A), pero antes de la ejecución de salida(B_B), donde B_A y B_B denotan los bloques de memoria intermedia en los que residen A y B. Al perderse el contenido de la memoria no se sabe la suerte de la transacción; así, podríamos invocar uno de los dos procedimientos posibles de recuperación.

- **Volver a ejecutar** T_i. Este procedimiento hará que el saldo de A se quede en 900 € en vez de en 950 €. De este modo el sistema entra en un estado inconsistente.
- **No volver a ejecutar** T_i. El estado actual del sistema otorga los valores de 950 € y 2.000 € para A y B respectivamente. Por tanto, el sistema entra en un estado inconsistente.

En cualquier caso se deja a la base de datos en un estado inconsistente y, por lo tanto, este esquema simple de recuperación de datos no funciona. El motivo de este mal funcionamiento es que se ha modificado la base de datos sin tener la seguridad de que la transacción se comprometa realmente. El objetivo es realizar todos los cambios inducidos por T_i o no llevar a cabo ninguno. Sin embargo, si T_i realiza varias modificaciones en la base de datos, pueden necesitarse varias operaciones de salida y puede ocurrir un fallo después de haberse concluido alguna de estas modificaciones, pero antes de haber terminado todas.

Para conseguir el objetivo de la atomicidad se debe efectuar primero la operación de salida de la información que describe las modificaciones en el almacenamiento estable sin modificar todavía la base de datos. Como se verá, este procedimiento permitirá realizar la salida de todas las modificaciones realizadas por una transacción comprometida aunque se produzcan fallos. Hay dos formas de ejecutar tales salidas; se estudian en los Apartados 17.4 y 17.5. En estos dos apartados supondremos que *las transacciones se ejecutan secuencialmente*, esto es, solamente una transacción está activa en cada momento. Se describirá la forma de manejar la ejecución concurrente de transacciones más adelante, en el Apartado 17.6.

17.4. RECUPERACIÓN BASADA EN EL REGISTRO HISTÓRICO

La estructura más ampliamente utilizada para guardar las modificaciones de una base de datos es el **registro histórico**. El registro histórico es una secuencia de **registros** que mantiene un registro de todas las actividades de actualización de la base de datos. Existen varios tipos de registros del registro histórico. Un **registro de actualización del registro histórico** describe una única escritura en la base de datos y tiene los siguientes campos:

- El **identificador de la transacción** es un identificador único de la transacción que realiza la operación escribir.
- El **identificador del elemento de datos** es un identificador único del elemento de datos que se escribe. Normalmente suele coincidir con la ubicación del elemento de datos en el disco.
- El **valor anterior** es el valor que tenía el elemento de datos antes de la escritura.
- El **valor nuevo** es el valor que tendrá el elemento de datos después de la escritura.

Existen otros registros del registro histórico especiales para registrar sucesos significativos durante el procesamiento de una transacción, tales como el comienzo de una transacción y el éxito o aborto de la misma. Denotaremos como sigue los diferentes tipos de registros del registro histórico:

- <T_i iniciada>. La transacción T_i ha comenzado.
- <T_i, X_j, V_1, V_2>. La transacción T_i ha realizado una escritura sobre el elemento de datos X_j. X_j tenía el valor V_1 antes de la escritura y tendrá el valor V_2 después de la escritura.
- <T_i comprometida>. La transacción T_i se ha comprometido.
- <T_i abortada>. La transacción T_i ha sido abortada.

Cuando una transacción realiza una escritura es fundamental que se cree el registro del registro histórico correspondiente a esa escritura antes de modificar la base de datos. Una vez que el registro del registro histórico existe, se puede realizar la salida de la modificación a la base de datos si se desea. Además, es posible *deshacer* una modificación que ya haya salido a la base de datos. Se deshará utilizando el campo valor-anterior de los registros del registro histórico.

Para que los registros del registro histórico sean útiles para recuperarse frente a errores del disco o del sistema, el registro histórico debe residir en almacenamiento estable. Por ahora supóngase que cada registro del registro histórico se escribe, tan pronto como se crea, al final del registro histórico en almacenamiento estable. En el Apartado 17.7 se verán las condiciones necesarias para poder relajar este requisito de forma segura de modo que se reduzca la sobrecarga impuesta por el registro histórico. En los Apartados 17.4.1 y 17.4.2 se presentarán dos técnicas de utilización del registro histórico para garantizar la atomicidad frente a fallos. Obsérvese que en el registro histórico se tiene constancia de todas las actividades de la base de datos. Como consecuencia, el tamaño de los datos almacenados en el registro histórico puede llegar a ser extremadamente grande. En el Apartado 17.4.3 se mostrará bajo qué condiciones se puede borrar información del registro histórico de manera segura.

17.4.1. Modificación diferida de la base de datos

La **técnica de la modificación diferida** garantiza la atomicidad de las transacciones mediante el almacenamiento de todas las modificaciones de la base de datos en el registro histórico, pero retardando la ejecución de todas las operaciones escribir de una transacción hasta que la transacción se compromete parcialmente. Recuérdese que se dice que una transacción se compromete parcialmente una vez que se ejecuta la acción final de la transacción. En la versión de la técnica de modificación diferida que se describe en este apartado se supone que las transacciones se ejecutan secuencialmente.

Cuando una transacción se compromete parcialmente, la información del registro histórico asociada a esa transacción se utiliza para la ejecución de las escrituras diferidas. Si el sistema cae antes de que la transacción complete su ejecución o si la transacción aborta, la información del registro histórico simplemente se ignora.

La ejecución de una transacción T_i opera de esta manera: antes de que T_i comience su ejecución se escribe en el registro histórico un registro <T_i iniciada>. Una operación escribir(X) realizada por T_i se traduce en la escritura de un nuevo registro en el registro histórico. Finalmente, cuando T_i se ha comprometido parcialmente, se escribe en el registro histórico un registro <T_i comprometida>.

Cuando T_i se compromete parcialmente, los registros asociados a ella en el registro histórico se utilizan para la ejecución de las escrituras diferidas. Como puede ocurrir un fallo mientras se lleva a cabo esta actualización, hay que asegurarse de que, antes del comienzo de estas actualizaciones, todos los registros del registro histórico se guardan en almacenamiento estable. Una vez que se ha hecho esto, la actualización real tiene lugar y la transacción pasa al estado comprometido.

Obsérvese que la técnica de modificación diferida sólo requiere el nuevo valor de los elementos de datos. Así, se puede simplificar la estructura general de los

registros de actualización del registro histórico que se vieron en el apartado anterior omitiendo el campo para el valor anterior.

Para ilustrar esto reconsidérese el sistema bancario simplificado. Sea T_0 una transacción que transfiere 50 € desde la cuenta A a la cuenta B. Esta transacción se define de la manera siguiente:

T_0 : leer(A)
A := A – 50
escribir(A)
leer(B)
B := B + 50
escribir(B)

Sea T_1 una transacción que retira 100 € de la cuenta C. Esta transacción se define como

T_1 : leer(C)
C := C – 100
escribir(C)

Supongamos que estas transacciones se ejecutan secuencialmente, primero T_0 y después T_1, y que los saldos de las cuentas A, B y C antes de producirse la ejecución eran 1.000, 2.000 y 700 € respectivamente. El fragmento del registro histórico que contiene la información relevante sobre estas dos transacciones se muestra en la Figura 17.2.

Las salidas reales que se producen en el sistema de bases de datos y en el registro histórico como consecuencia de la ejecución de T_0 y T_1 pueden seguir distintas ordenaciones. Una ordenación posible se presenta en la Figura 17.3. Nótese que el valor de A se cambia en la base de datos sólo después de que el registro <T_0, A, 950> se haya introducido en el registro histórico.

Mediante la utilización del registro histórico, el sistema puede manejar cualquier fallo que conduzca a la pérdida de información en el almacenamiento volátil. El esquema de recuperación usa el siguiente procedimiento de recuperación:

- rehacer(T_i) fija el valor de todos los elementos de datos actualizados por la transacción T_i a los valores nuevos.

El conjunto de elementos de datos actualizados por T_i y sus respectivos nuevos valores se encuentran en el registro histórico.

<T_0 iniciada>
<T_0, A, 950>
<T_0, B, 450>
<T_0 comprometida>
<T_1 iniciada>
<T_1, C, 1100>
<T_1 comprometida>

FIGURA 17.2. Fragmento del registro histórico de la base de datos correspondiente a T_0 y T_1.

Registro histórico	Base de datos
<T_0 iniciada>	
<T_0, A, 950>	
<T_0, B, 2050>	
<T_0 comprometida>	
	A = 950
	B = 2050
<T_1 iniciada>	
<T_1, C, 600>	
<T_1 comprometida>	
	C = 600

FIGURA 17.3. Estado del registro histórico y de la base de datos correspondiente a T_0 y T_1.

La operación rehacer debe ser **idempotente**, esto es, el resultado de ejecutarla varias veces debe ser equivalente al resultado de ejecutarla una sola vez. Esta característica es fundamental para garantizar un correcto comportamiento incluso si el fallo se produce durante el proceso de recuperación.

Después de ocurrir un fallo, el subsistema de recuperación consulta el registro histórico para determinar las transacciones que deben rehacerse. Una transacción T_i debe rehacerse si y sólo si el registro histórico contiene los registros <T_i iniciada> y <T_i comprometida>. Así, si el sistema cae después de que la transacción complete su ejecución, la información en el registro histórico se utiliza para restituir el sistema a un estado consistente anterior.

Para ilustrar esto volvamos a considerar el ejemplo bancario con la ejecución ordenada de las transacciones T_0 y T_1, primero T_0 y después T_1. En la Figura 17.2 se muestra el registro histórico que resulta de la ejecución completa de T_0 y T_1. Supóngase que el sistema cae antes de completarse las transacciones para poder ver el modo en que la técnica de recuperación lleva a la base de datos a un estado consistente. Supóngase que la caída ocurre justo después de haber escrito en almacenamiento estable el registro del registro histórico para el paso

escribir(B)

de la transacción T_0. El contenido del registro histórico en el momento de la caída puede verse en la Figura 17.4a. Cuando el sistema vuelve a funcionar no es necesario llevar a cabo ninguna acción rehacer, ya que no aparece ningún registro de compromiso en el registro histórico. Los saldos de las cuentas A y B siguen siendo de 1.000 y 2.000 € respectivamente. Pueden borrarse del registro histórico los registros de la transacción incompleta T_0.

Supóngase ahora que la caída sucede justo después de haber escrito en almacenamiento estable el registro del registro histórico para el paso

escribir(C)

de la transacción T_1. En este caso, el contenido del registro histórico en el momento de la caída puede verse en la

(a)	(b)	(c)
<T_0 iniciada>	<T_0 iniciada>	<T_0 iniciada>
<T_0, A, 950>	<T_0, A, 950>	<T_0, A, 950>
<T_0, B, 2050>	<T_0, B, 2050>	<T_0, B, 2050>
	<T_0 comprometida>	<T_0 comprometida>
	<T_1 iniciada>	<T_1 iniciada>
	<T_1, C, 600>	<T_1, C, 600>
		<T_0 comprometida>

FIGURA 17.4. El mismo registro histórico que el de la Figura 17.3 en tres momentos distintos.

Figura 17.4b. Cuando el sistema vuelve a funcionar se realiza la operación rehacer(T_0), ya que el registro

<T_0 comprometida>

aparece en el registro histórico en el disco. Después de la ejecución de esta operación, los saldos de las cuentas A y B son de 950 y 2.050 € respectivamente. El saldo de la cuenta C se mantiene en 700 €. Igual que antes, pueden borrarse del registro histórico los registros de la transacción incompleta T_1.

Por último, supóngase que la caída ocurre justo después de haber escrito en almacenamiento estable el registro del registro histórico

<T_1 comprometida>

El contenido del registro histórico en el momento de la caída se muestra en la Figura 17.4c. Cuando el sistema vuelve a funcionar, hay dos registros comprometida en el registro: uno para T_0 y otro para T_1. Así pues, deben realizarse las operaciones rehacer(T_0) y rehacer(T_1). Después de la ejecución de estas operaciones, los saldos de las cuentas A, B y C son de 950, 2.050 y 600 € respectivamente.

Considérese finalmente un caso en el que tiene lugar una segunda caída del sistema durante la recuperación de la primera. Deben hacerse algunos cambios en la base de datos como consecuencia de la ejecución de las operaciones rehacer, pero no se han realizado todos los cambios. Cuando el sistema vuelve a funcionar después de la segunda caída, la recuperación procede exactamente igual que en los ejemplos anteriores. Para cada

<T_i comprometida>

que se encuentre en el registro histórico, se lanza la operación rehacer(T_i). En otras palabras, las acciones de recuperación se vuelven a reanudar desde el principio. Como rehacer escribe los valores en la base de datos independientemente de los datos que haya actualmente en la base de datos, el resultado de un segundo intento acabado con éxito en la ejecución de rehacer es el mismo que si rehacer hubiera acabado con éxito la primera vez.

17.4.2. Modificación inmediata de la base de datos

La **técnica de modificación inmediata** permite realizar la salida de las modificaciones de la base de datos a la propia base de datos mientras que la transacción está todavía en estado activo. Las modificaciones de datos escritas por transacciones activas se denominan **modificaciones no comprometidas**. En caso de una caída o de un fallo en la transacción, el sistema debe utilizar el campo para el valor anterior de los registros del registro histórico descritos en el Apartado 17.4 para restaurar los elementos de datos modificados a los valores que tuvieran antes de comenzar la transacción. Esta restauración se lleva a cabo mediante la operación deshacer descrita a continuación.

Antes de comenzar la ejecución de una transacción T_i, se escribe en el registro histórico el registro <T_i iniciada>. Durante su ejecución, cualquier operación escribir(X) realizada por T_i, es *precedida* por la escritura en el registro histórico de un registro actualizado apropiado. Cuando T_i se compromete parcialmente se escribe en el registro histórico el registro <T_i comprometida>.

Como la información del registro histórico se utiliza para reconstruir el estado de la base de datos, la actualización real de la base de datos no puede permitirse antes de que el registro del registro histórico correspondiente se haya escrito en almacenamiento estable. Por lo tanto, es necesario que antes de la ejecución de una operación de salida(B), se escriban en almacenamiento estable los registros del registro histórico correspondientes a B. Esto volverá a tratarse en el Apartado 17.7.

Para ilustrarlo considérese de nuevo el sistema bancario simplificado con la ejecución ordenada de las transacciones T_0 y T_1, primero T_0 y después T_1. Las líneas del registro histórico que contienen la información relevante concerniente a estas dos transacciones se muestran en la Figura 17.5.

En la Figura 17.6 se describe una posible ordenación de las salidas reales que se producen en el sistema de bases de datos y en el registro histórico como consecuencia de la ejecución de T_0 y T_1. Nótese que esta ordenación no podría obtenerse con la técnica de modificación diferida que se vio en el Apartado 17.4.1.

Mediante la utilización del registro histórico, el sistema puede manejar cualquier fallo que no genere una

<T_0 iniciada>
<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>
<T_0 comprometida>
<T_1 iniciada>
<T_1, C, 700, 600>
<T_1 comprometida>

FIGURA 17.5. Fragmento del registro histórico del sistema correspondiente a T_0 y T_1.

Registro histórico	Base de datos
<T_0 iniciada>	
<T_0, A, 1000, 950>	
<T_0, B, 2000, 2050>	
	A = 950
	B = 2050
<T_0 comprometida>	
<T_1 iniciada>	
<T_1, C, 700, 600>	
	C = 600
<T_1 comprometida>	

FIGURA 17.6. Estado del registro histórico y de la base de datos correspondiente a T_0 y T_1.

pérdida de información en el almacenamiento no volátil. El esquema de recuperación usa dos procedimientos de recuperación:

- deshacer(T_i) restaura el valor de todos los elementos de datos actualizados por la transacción T_i a los valores anteriores.
- rehacer(T_i) fija el valor de todos los elementos de datos actualizados por la transacción T_i a los nuevos valores.

El conjunto de elementos de datos actualizados por T_i y sus respectivos anteriores y nuevos valores se encuentran en el registro histórico.

Las operaciones deshacer y rehacer deben ser idempotentes para garantizar un comportamiento correcto incluso en el caso de que el fallo se produzca durante el proceso de recuperación.

Después de haberse producido un fallo, el esquema de recuperación consulta el registro histórico para determinar las transacciones que deben rehacerse y las que deben deshacerse.

- Una transacción T_i debe deshacerse si el registro histórico contiene el registro <T_i iniciada>, pero no contiene el registro <T_i comprometida>.
- Una transacción T_i debe rehacerse si el registro histórico contiene los registros <T_i iniciada> y <T_i comprometida>.

Para ilustrarlo, considérese de nuevo el sistema bancario con la ejecución ordenada de las transacciones T_0 y T_1, primero T_0 y después T_1. Supóngase que el sistema cae antes de completarse las transacciones. Se considerarán tres casos. El estado del registro histórico para cada uno de ellos se muestra en la Figura 17.7.

En primer lugar supóngase que la caída ocurre justo después de haber escrito en almacenamiento estable el registro del registro histórico para el paso

escribir(B)

de la transacción T_0 (Figura 17.7a). Cuando el sistema vuelve a funcionar encuentra en el registro histórico el registro <T_0 iniciada>, pero no su correspondiente <T_0 comprometida>. Por lo tanto, la transacción T_0 debe deshacerse y se ejecutaría deshacer(T_0). Como resultado de esta operación, los saldos de las cuentas A y B (en el disco) se restituirían a 1.000 y 2.000 € respectivamente.

Supóngase ahora que la caída sucede justo después de haber escrito en almacenamiento estable el registro del registro histórico para el paso

escribir(C)

de la transacción T_1 (Figura 17.7b). Cuando el sistema vuelve a funcionar, es necesario llevar a cabo dos acciones de recuperación. La operación deshacer(T_1) debe ejecutarse porque en el registro histórico aparece el registro <T_1 iniciada>, pero no aparece <T_1 comprometida>. La operación rehacer(T_0) debe ejecutarse porque el registro histórico contiene los registros <T_0 iniciada> y <T_0 comprometida>. Al final del procedimiento de recuperación, los saldos de las cuentas A, B y C son de

<T_0 iniciada>	<T_0 iniciada>	<T_0 iniciada>
<T_0, A, 1000,950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000,2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 comprometida>	<T_0 comprometida>
	<T_1 iniciada>	<T_1 iniciada>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 comprometida>
(a)	(b)	(c)

FIGURA 17.7. El mismo registro histórico mostrado en tres momentos distintos.

950, 2.050 y 700 € respectivamente. Nótese que la operación deshacer(T_1) se ejecuta antes que rehacer(T_0). En este ejemplo, el resultado sería el mismo si se cambiara el orden. Sin embargo, el hecho de realizar primero las operaciones deshacer y luego las operaciones rehacer es importante para el algoritmo de recuperación que se describe en el Apartado 17.6.

Por último, supóngase que la caída tiene lugar justo después de haber escrito en almacenamiento estable el registro del registro histórico

<T_1 comprometida>

(Figura 17.7c). Cuando el sistema vuelve a funcionar, deben rehacerse tanto T_0 como T_1 ya que se encuentran en el registro histórico los registros <T_0 iniciada> y <T_0 comprometida>, así como los registros <T_1 iniciada> y <T_1 comprometida>. Los saldos de las cuentas A, B y C después de la ejecución de los procedimientos de recuperación rehacer(T_0) y rehacer(T_1) se sitúan en 950, 2.050 y 600 € respectivamente.

17.4.3. Puntos de revisión

Cuando ocurre un fallo en el sistema se debe consultar el registro histórico para determinar las transacciones que deben rehacerse y las que deben deshacerse. En principio es necesario recorrer completamente el registro histórico para hallar esta información. En este enfoque hay dos inconvenientes principales:

1. El proceso de búsqueda consume tiempo.
2. La mayoría de las transacciones que deben rehacerse de acuerdo con el algoritmo ya tienen escritas sus actualizaciones en la base de datos. Aunque el hecho de volver a ejecutar estas transacciones no produzca resultados erróneos, sí repercutirá en un aumento del tiempo de ejecución del proceso de recuperación.

Para reducir este tipo de sobrecarga se introducen los puntos de revisión. Durante la ejecución, el sistema actualiza el registro histórico utilizando una de las dos técnicas que se describen en los Apartados 17.4.1 y 17.4.2. Además, el sistema realiza periódicamente **puntos de revisión**, en los cuales tiene lugar la siguiente secuencia de acciones:

1. Escritura en almacenamiento estable de todos los registros del registro histórico que residan en ese momento en memoria principal.
2. Escritura en disco de todos los bloques de memoria intermedia que se hayan modificado.
3. Escritura en almacenamiento estable de un registro del registro histórico <revisión>.

Mientras se lleva a cabo un punto de revisión no se permite que ninguna transacción realice acciones de actualización, tales como escribir en un bloque de memoria intermedia o escribir un registro del registro histórico.

La presencia de un registro <revisión> en el registro histórico permite que el sistema pueda hacer más eficiente su procedimiento de recuperación. Considérese una transacción T_i que se comprometió antes del punto de revisión. Para esa transacción el registro <T_i comprometida> aparece en el registro histórico antes que el registro <revisión>. Todas las modificaciones sobre la base de datos hechas por T_i se deben haber escrito en la base de datos antes del punto de revisión o formando parte del propio punto de revisión. Así, en el momento de la recuperación, no es necesario ejecutar una operación rehacer sobre T_i.

Esta observación permite perfeccionar los esquemas anteriores de recuperación (sigue siendo válido el supuesto de que las transacciones se ejecutan secuencialmente). Cuando se produce un fallo, el esquema de recuperación examina el registro histórico para determinar la última transacción T_i que comenzó su ejecución antes de que tuviera lugar el último punto de revisión. Para encontrar una transacción de este tipo se recorre el registro histórico hacia atrás, esto es, se empieza a buscar por el final del registro histórico hasta que se encuentra el primer registro <revisión> (como se va recorriendo el registro histórico hacia atrás, el registro encontrado corresponde al último registro <revisión> del registro histórico); después se continúa la búsqueda hacia atrás hasta que se encuentra el siguiente registro <T_i iniciada>. Este registro identifica a la transacción T_i.

Una vez que ha sido identificada la transacción T_i sólo es necesario aplicar las operaciones rehacer y deshacer a la transacción T_i y a las transacciones T_j que comenzaron su ejecución después que T_i. Sea T este conjunto de transacciones. Puede ignorarse el resto del registro histórico (la parte del principio) y puede borrarse cuando se desee. El conjunto exacto de operaciones de recuperación que han de llevarse a cabo depende de si se está usando la técnica de modificación inmediata o la de modificación diferida. Si se emplea la técnica de modificación inmediata, las operaciones de recuperación deben ser las siguientes:

- Ejecutar deshacer(T_k) para todas las transacciones T_k de T para las que no exista un registro <T_k comprometida> en el registro histórico.
- Ejecutar rehacer(T_k) para todas las transacciones T_k de T para las que aparece un registro <T_k comprometida> en el registro histórico.

Obviamente no es necesario aplicar la operación rehacer cuando se está utilizando la técnica de modificación diferida.

Como ejemplo considérese el conjunto de transacciones $\{T_0, T_1, \ldots, T_{100}\}$ de modo que su ejecución se

produce en el orden determinado por los subíndices. Supóngase que el último punto de revisión tiene lugar durante la ejecución de la transacción T_{67}. Así, durante el esquema de recuperación, sólo deben considerarse las transacciones T_{67}, T_{68} ,…, T_{100}. Será necesario rehacer cada una de ellas si éstas se comprometieron y será necesario deshacerlas en caso contrario.

En el Apartado 17.6.3 se estudia una extensión de la técnica de puntos de revisión para el procesamiento concurrente de transacciones.

17.5. PAGINACIÓN EN LA SOMBRA

La paginación en la sombra es una técnica de recuperación alternativa a las basadas en registro histórico. La técnica de paginación en la sombra es esencialmente la técnica de copia en la sombra que se vio en el Apartado 13.3 con algunas mejoras. Bajo ciertas circunstancias la paginación en la sombra puede requerir menos accesos al disco que los métodos basados en registro histórico que se presentaron anteriormente. No obstante, como se verá, existen algunos inconvenientes en el enfoque de la paginación en la sombra. Por ejemplo, es difícil extender la paginación en la sombra para permitir que varias transacciones puedan ejecutarse concurrentemente.

Igual que antes, la base de datos se divide en un número determinado de bloques de longitud fija a los que se denominará **páginas**. Debido a que se va a utilizar un esquema de paginación para la gestión de la memoria, se ha tomado prestado de los sistemas operativos el término *página*. Supóngase que hay n páginas numeradas desde 1 hasta n (en la práctica, n puede ser del orden de cientos de miles). No es necesario almacenar en disco estas páginas en un orden determinado (como se vio en el Capítulo 11 hay muchas razones por las que esto es así). Sin embargo, dado un cierto i, debe existir una manera de localizar la página i-ésima de la base de datos. Para este propósito se usará una **tabla de páginas** como se muestra en la Figura 17.8. La tabla de páginas tiene n entradas, una para cada página de la base de datos. Cada entrada contiene un puntero a una página en el disco. La primera entrada contiene un puntero a la primera página de la base de datos, la segunda entrada apunta a la segunda página y así sucesivamente. En el ejemplo de la Figura 17.8 se muestra que el orden lógico de las páginas de la base de datos no tiene por qué coincidir con el orden físico en el que están almacenadas en el disco.

La idea principal que subyace tras la paginación en la sombra es la de mantener *dos* tablas de páginas durante la vida de una transacción: la **tabla de páginas actual** y la **tabla de páginas sombra**. Estas dos tablas son idénticas cuando comienza una transacción. Mientras dura la transacción no se altera el contenido de la tabla de páginas sombra. Cuando una transacción realiza una operación escribir, la tabla actual de páginas puede sufrir algún cambio. Todas las operaciones entrada y salida usan la tabla actual de páginas para encontrar las páginas de la base de datos en el disco.

Supóngase que la transacción realiza una operación escribir(X) y que X pertenece a la página i-ésima. La operación escribir se ejecutaría como sigue:

1. Ejecutar entrada(X) si la página i-ésima (es decir, la página en la que se encuentra X) no está todavía en memoria principal.
2. Si es la primera escritura que esta transacción realiza sobre la página i-ésima, modificar la tabla actual de páginas siguiendo estos pasos:
   a. Encontrar una página en el disco que no se haya utilizado. Normalmente, como vimos en el Capítulo 11, el sistema de bases de datos tiene acceso a una lista de páginas no utilizadas (libres).
   b. Borrar la página encontrada en el paso anterior de la lista de marcos de página libres.
   c. Modificar la tabla actual de páginas de modo que la entrada i-ésima apunte a la página encontrada en el paso 2a.
3. Asignar el valor de x_j a X en la página de la memoria intermedia.

Compárense las acciones que se acaban de ver para realizar una operación escribir con las que se describieron en el Apartado 17.2. La única diferencia es que se ha añadido un nuevo paso. Los pasos 1 y 3 se corresponden con los pasos 1 y 2 del Apartado 17.2. El paso nuevo, el segundo, manipula la tabla actual de páginas. En la figura 17.9 se muestran las tablas actual y en la sombra para una transacción que está realizando una escritura en la cuarta página de una base de datos que tiene 10 páginas.

Intuitivamente, el enfoque de la paginación en la sombra para recuperación se basa en almacenar la tabla de páginas sombra en almacenamiento no volátil, de modo que puede recuperarse el estado de la base de datos antes de la ejecución de una transacción en caso de producirse una caída del sistema o de que se aborta-se la transacción. La tabla actual de páginas se escribe en almacenamiento no volátil cuando la transacción se compromete. Entonces, la tabla actual de páginas se convierte en la nueva tabla de páginas sombra y se concede el permiso para la ejecución de la siguiente transacción. El hecho de que la tabla de páginas sombra se guarde en almacenamiento no volátil tiene gran impor-

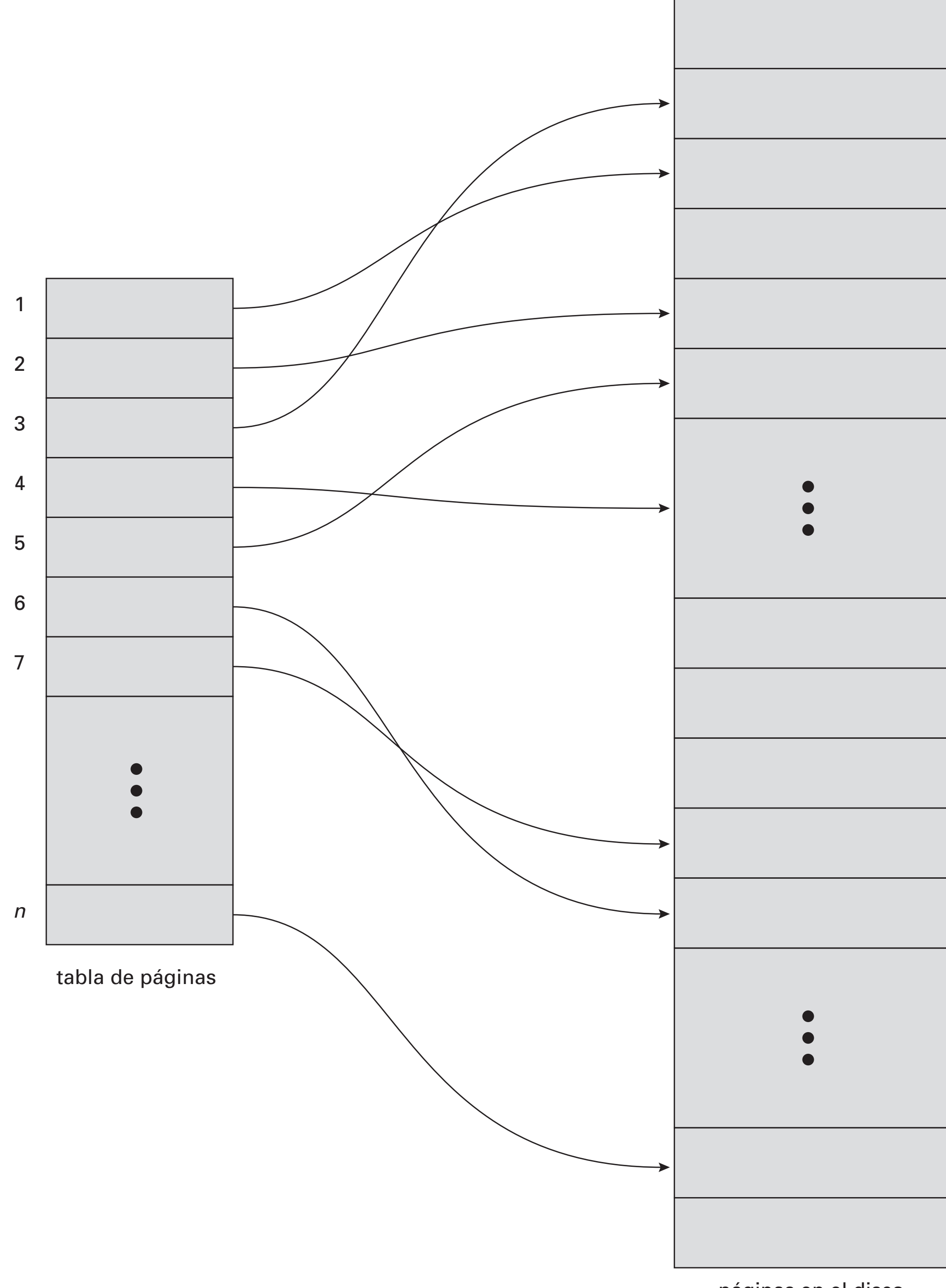


FIGURA 17.8. Ejemplo de tabla de páginas.

tancia debido a que proporciona el único medio para encontrar las páginas de la base de datos. La tabla actual de páginas puede almacenarse en memoria principal (soporte de almacenamiento volátil). Como el sistema utiliza la tabla de páginas sombra para recuperarse, no importa si la tabla actual de páginas se pierde en una caída del sistema.

Es necesario que, después de una caída del sistema, sea posible encontrar en el disco la tabla de páginas sombra para poder realizar con éxito el proceso de recuperación. Una manera sencilla de encontrarla es elegir un lugar determinado del almacenamiento estable para albergar las direcciones del disco en las que se encuentra la tabla de páginas sombra. Cuando el sistema vuelve a funcionar después de una caída, se copia la tabla de páginas sombra en memoria principal y se utiliza para el procesamiento de las siguientes transacciones. Está garantizado, por la definición de la operación escribir, que la tabla de páginas sombra apuntará a las páginas de la base de datos correspondientes al estado anterior a cualquier transacción que estuviera activa en el momento de la caída. De esta manera, los abortos son

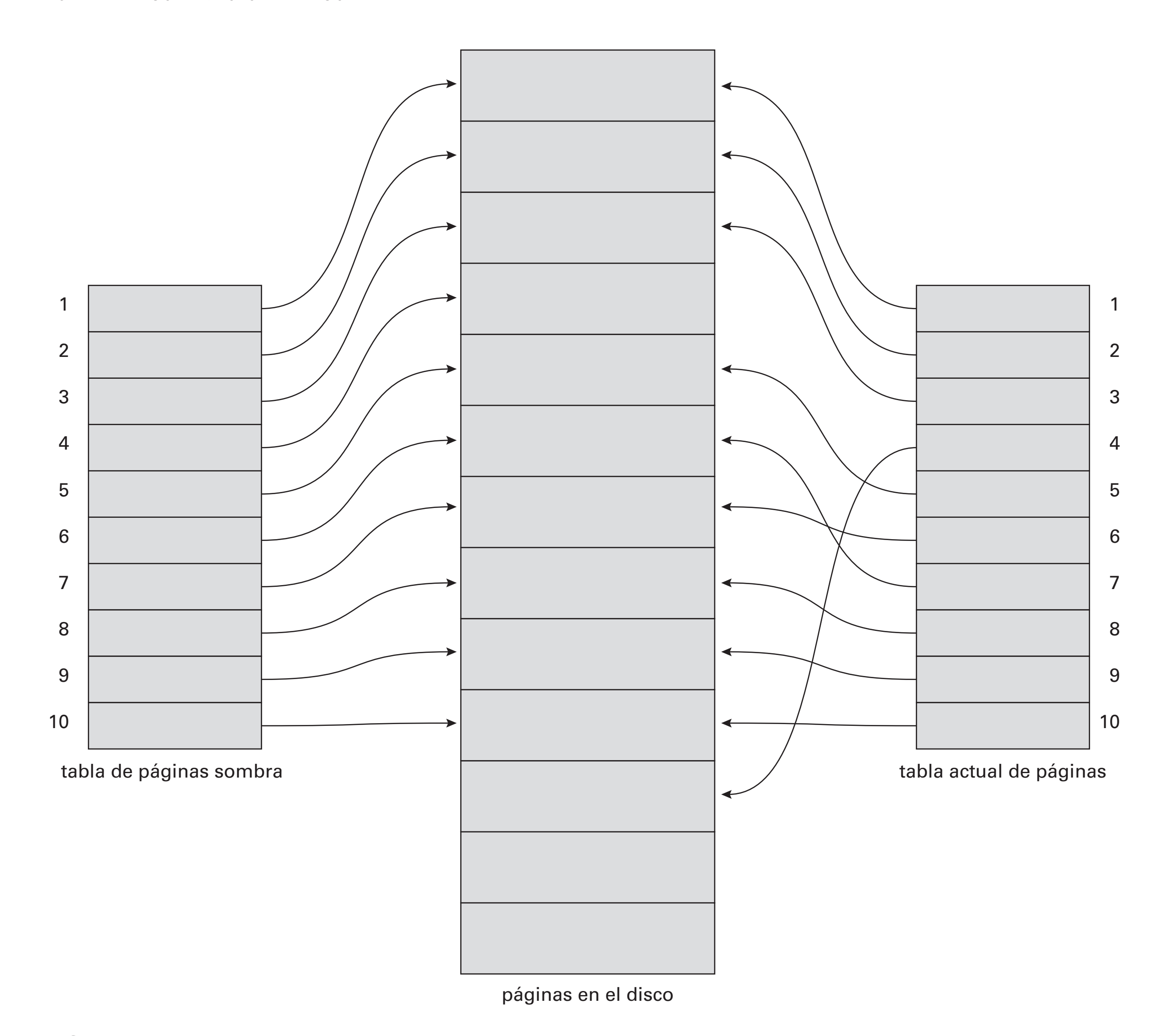


FIGURA 17.9. Tablas de páginas actual y en la sombra.

automáticos. A diferencia de los esquemas basados en registro histórico, no es necesario realizar ninguna operación deshacer.

Para comprometer una transacción se deben seguir estos pasos:

1. Asegurarse de que se escriben en disco todas las páginas de la memoria intermedia que se encuentren en memoria principal y que hayan sido modificadas por la transacción. (Nótese que estas operaciones de escritura en disco no cambiarán las páginas de la base de datos que son apuntadas por las entradas de la tabla de páginas sombra.)
2. Escribir en disco la tabla actual de páginas. No se debe sobreescribir la tabla de páginas sombra pues puede ser necesaria para el proceso de recuperación si ocurriera una caída.
3. Escribir las direcciones del disco correspondientes a la tabla actual de páginas en el lugar del almacenamiento estable que contiene la dirección de la tabla de páginas sombra. Esta acción sobreescribe las direcciones de la anterior tabla de páginas sombra. Por lo tanto, la tabla actual es ahora la tabla en la sombra y la transacción se compromete.

Si se produce una caída del sistema antes de la finalización del paso 3, se vuelve al estado inmediatamente anterior a la ejecución de la transacción. Si la caída sucede después de completar el paso 3, se conservarán los efectos de la transacción; no es necesario realizar ninguna operación rehacer.

La paginación en la sombra presenta varias ventajas frente a las técnicas basadas en registro histórico. Se elimina la sobrecarga de escrituras del registro histórico y la recuperación es notablemente más rápida (ya que no son necesarias las operaciones rehacer ni deshacer). Sin embargo, la técnica de paginación en la sombra también tiene ciertos inconvenientes:

- **Sobrecarga en el compromiso.** Comprometer una sola transacción que utilice paginación en la sombra precisa la escritura de muchos bloques: los bloques de datos reales, la tabla actual de páginas y las direcciones del disco de la tabla actual de páginas. Los esquemas basados en registro histórico sólo necesitan escribir los registros del registro histórico, los cuales, para pequeñas transacciones típicas, caben en un solo bloque.

  La sobrecarga de la escritura de una tabla de páginas completa se puede reducir implementando la tabla de páginas como una estructura de árbol con las tablas de páginas en las hojas. A continuación se esboza esta técnica, dejando al lector que complete los detalles ausentes. Los nodos del árbol son páginas y tienen un alto grado de salida, como los árboles B^+. El árbol de la tabla de páginas actual es inicialmente el mismo que el árbol de la tablas de páginas sombra. Cuando se actualiza por primera vez una página, el sistema cambia la entrada de la tabla de páginas actual para que apunte a la copia de la página. Si la página hoja que contiene la entrada ya se ha copiado, el sistema lo actualiza directamente. En caso contrario, el sistema primero lo copia y actualiza la copia. A su vez, es necesario actualizar el padre de la página copiada para que apunte a la nueva copia, lo que se realiza aplicando el mismo procedimiento a su padre, copiándolo si aún no se hubiese hecho. El proceso de la copia se realiza hasta la raíz del árbol. Los cambios sólo se hacen sobre los nodos copiados, así que el árbol de la tabla de páginas no se modifica.

  La ventaja de la representación de árbol es que sólo es necesario copiar las páginas hoja que hayan sido actualizadas, y todos sus ascendientes en el árbol. El resto de partes del árbol se comparten entre la página de tablas sombra y la actual, y no es necesario copiarlas. La reducción de los costes de copia puede ser significativa en grandes bases de datos. Sin embargo, aún es necesario copiar varias páginas de la tabla de páginas para cada transacción, y los esquemas basados en registro histórico continúan siendo superiores siempre que las transacciones actualizan sólo pequeñas partes de la base de datos.

- **Fragmentación de datos.** En el Capítulo 11 se consideraban ciertas estrategias para asegurar la localidad, es decir, mantener físicamente cercanas en el disco las páginas de la base de datos que estaban relacionadas entre sí. Esta localidad permite transferencias de datos más veloces. La paginación en la sombra hace que las páginas de la base de datos cambien su ubicación en el disco siempre que se modifiquen. Como consecuencia, o se pierde la propiedad de localidad de las páginas o debe acudirse a esquemas de gestión del almacenamiento físico más complicados y con mayor sobrecarga (véanse las notas bibliográficas para obtener más información).

- **Recogida de basura.** Cada vez que se compromete una transacción, las páginas de la base de datos que contenían la versión anterior de los datos y que fueron modificadas por la transacción se hacen inaccesibles. En la Figura 17.9, cuando la transacción del ejemplo se comprometa, la página apuntada por la cuarta entrada de la tabla de páginas sombra pasará a ser inaccesible. Las páginas como ésa son consideradas como **basura** debido a que no forman parte del espacio libre y no contienen ninguna información útil. Puede producirse basura también como efecto lateral de las caídas. Es necesario encontrar periódicamente todas las páginas basura para añadirlas a la lista de páginas libres. Este proceso, denominado **recogida de basura**, añade al sistema sobrecarga y complejidad. Existen varios algoritmos estándar para la recogida de basura (véanse las referencias en las notas bibliográficas).

Además de los inconvenientes que se acaban de mencionar, la paginación en la sombra presenta más dificultades que las técnicas basadas en registro histórico para adaptarla a sistemas que permitan la ejecución concurrente de varias transacciones. En estos sistemas, aunque se utilice la paginación en la sombra, suele ser necesaria alguna técnica basada en registro histórico. El prototipo System R, por ejemplo, utiliza una combinación de paginación en la sombra y un esquema basado en registro histórico similar al presentado en el Apartado 17.4.2. Como se verá en el Apartado 17.6, es relativamente sencillo extender los esquemas de recuperación basados en registro histórico para permitir transacciones concurrentes. Por todas estas razones no está muy extendido el uso de la paginación en la sombra.

17.6. TRANSACCIONES CONCURRENTES Y RECUPERACIÓN

Hasta ahora se ha tratado la recuperación en un entorno en el que se ejecutaba una sola transacción en cada instante. Ahora se verá cómo modificar y extender el esquema de recuperación basado en registro histórico para permitir la ejecución concurrente de varias transacciones. El sistema sigue teniendo una única memoria intermedia de disco y un único registro histórico independientemente del número de transacciones concurrentes. Todas las transacciones comparten los bloques de la memoria intermedia. Se permiten actualiza-

ciones inmediatas y que un bloque de la memoria intermedia tenga elementos de datos que hayan sido modificados por una o más transacciones.

17.6.1. Interacción con el control de concurrencia

El esquema recuperación depende en gran medida del esquema de control de concurrencia que se use. Para retroceder los efectos de una transacción fallida deben deshacerse las modificaciones realizadas por esa transacción. Supóngase que se debe retroceder una transacción T_0 y que un dato Q, que fue modificado por T_0, tiene que recuperar su antiguo valor. Si se está usando un esquema basado en registro histórico para la recuperación, es posible restablecer el valor de Q utilizando la información contenida en el registro histórico. Supóngase ahora que una segunda transacción T_1 realiza una nueva modificación sobre Q *antes* de retroceder T_0. En este caso, al retroceder T_0, se perdería la modificación realizada por T_1.

Es necesario, por tanto, que si una transacción T modifica el valor de un elemento de datos Q, ninguna otra transacción pueda modificar el mismo elemento de datos hasta que T se haya comprometido o se haya retrocedido. Este requisito puede satisfacerse fácilmente utilizando bloqueo estricto de dos fases, esto es, bloqueo de dos fases manteniendo bloqueos exclusivos hasta el final de la transacción.

17.6.2. Retroceso de transacciones

Se utiliza el registro histórico para retroceder una transacción T_i fallida. El registro histórico se explora hacia atrás; para cada registro del registro histórico de la forma <T_i, X_j, V_1, V_2>, se restablece el valor del elemento de datos X_j con su valor anterior: V_1. La exploración del registro histórico termina cuando se encuentra el registro <T_i iniciada>.

Es importante el hecho de recorrer el registro histórico empezando por el final, ya que una transacción puede haber actualizado más de una vez el valor de un elemento de datos. Como ejemplo, considérese este par de registros:

<T_i, A, 10, 20>
<T_i, A, 20, 30>

Estos registros del registro histórico representan una modificación del elemento de datos A por parte de la transacción T_i, seguida de otra modificación de A hecha también por T_i. Al recorrer el registro histórico al revés se establece correctamente el valor de A como 10. Si el registro histórico se recorriera hacia delante, A tomaría como valor 20, lo cual es incorrecto.

Si para el control de concurrencia se utiliza el bloqueo estricto de dos fases, los bloqueos llevados a cabo por una transacción T sólo pueden ser desbloqueados después de que la transacción se haya retrocedido según se acaba de describir. Una vez que T (que se está retrocediendo) haya actualizado un elemento de datos, ninguna otra transacción podría haber actualizado el mismo elemento de datos debido a los requisitos del control de concurrencia que se mencionaron en el apartado 17.6.1. Así pues, la restitución del valor anterior de un elemento de datos no borrará los efectos de otra transacción.

17.6.3. Puntos de revisión

En el Apartado 17.4.3 se usaban los puntos de revisión para reducir el número de registros del registro histórico que debían ser examinados cuando el sistema se recuperaba de una caída. Como se asumía que no existía la concurrencia, durante la recuperación era necesario considerar solamente las siguientes transacciones:

- Las transacciones que comenzaron después del último punto de revisión
- La única transacción, si la había, que estaba activa en el momento de grabarse el último punto de revisión

Cuando las transacciones pueden ejecutarse concurrentemente, la situación se torna más complicada ya que varias transacciones pueden estar activas en el momento en que se produce el último punto de revisión.

En un sistema de procesamiento de transacciones concurrente es necesario que el registro del registro histórico correspondiente a un punto de revisión sea de la forma <revisión L>, donde L es una lista con las transacciones activas en el momento del punto de revisión. De nuevo se supone que, mientras que se realiza el punto de revisión, las transacciones no efectúan modificaciones ni sobre los bloques de la memoria intermedia ni sobre el registro histórico.

El requisito de que las transacciones no puedan realizar modificaciones sobre los bloques de la memoria intermedia ni sobre el registro histórico durante un punto de revisión puede resultar molesto, ya que el procesamiento de transacciones tendrá que parar durante la ejecución de un punto de revisión. Un punto de revisión durante el cual se permite que las transacciones realicen modificaciones incluso mientras los bloques de memoria intermedia se están guardando en disco, se denomina **punto de revisión difuso**. En el Apartado 17.9.5 se describen esquemas de revisión difusa.

17.6.4. Recuperación al reiniciar

El sistema construye dos listas cuando se recupera de una caída: la *lista-deshacer*, que consta de las transacciones que han de deshacerse, y la *lista-rehacer*, que está formada por las transacciones que deben rehacerse.

Estas dos listas se construyen durante la recuperación de la siguiente manera. Al principio ambas están vacías. Luego se recorre el registro histórico hacia atrás examinando cada registro hasta que se encuentra el primer registro <revisión>:

- Para cada registro encontrado de la forma <T_i comprometida>, se añade T_i a la *lista-rehacer*.
- Para cada registro encontrado de la forma <T_i iniciada>, si T_i no está en la *lista-rehacer*, entonces se añade T_i a la *lista-deshacer*.

Una vez que se han examinado los registros apropiados del registro histórico, se atiende al contenido de la lista L en el registro punto de revisión. Para cada transacción T_i en L, si T_i no está en la *lista-rehacer*, entonces se añade T_i a la *lista-deshacer*.

Cuando se terminan la *lista-rehacer* y la *lista-deshacer*, el proceso de recuperación procede de la siguiente manera:

1. Se recorre de nuevo el registro histórico hacia atrás comenzando en el último registro y se realiza una operación deshacer por cada registro del registro histórico que pertenezca a una transacción T_i de la *lista-deshacer*. En esta fase se ignoran los registros del registro histórico concernientes a transacciones de la *lista-rehacer*. El recorrido del registro histórico termina cuando se encuentran registros <T_i iniciada> para cada transacción T_i de la *lista-deshacer*.
2. Se localiza el último registro <revisión L> del registro histórico. Nótese que este paso puede necesitar de un recorrido del registro histórico hacia delante si el registro punto de revisión quedó atrás en el paso 1.
3. Se recorre el registro histórico hacia delante desde el último registro <revisión L> y se realiza una operación rehacer por cada registro del registro histórico que pertenezca a una transacción T_i de la *lista-rehacer*. En esta fase se ignoran los registros del registro histórico concernientes a transacciones de la *lista-deshacer*.

Es importante procesar el registro histórico hacia atrás en el paso 1 para garantizar que el estado resultante de la base de datos sea correcto.

Después de haber deshecho todas las transacciones de la *lista-deshacer* se rehacen aquellas transacciones que pertenezcan a la *lista-rehacer*. En este caso es importante procesar el registro histórico hacia delante. Cuando se ha completado el proceso de recuperación, se continúa con el procesamiento normal de las transacciones.

Es importante el hecho de deshacer las transacciones de la *lista-deshacer* antes de rehacer las transacciones de la *lista-rehacer* al utilizar los pasos 1 a 3 del algoritmo anterior. El siguiente problema podría ocurrir de no hacerse así. Supóngase que el elemento de datos A vale inicialmente 10. Supóngase también que una transacción T_i modifica el valor de A situándolo en 20 y aborta a continuación; el retroceso de la transacción devolvería a A el valor 10. Supóngase que otra transacción T_j cambia entonces a 30 el valor de A, se compromete y, seguidamente, el sistema cae. El estado del registro histórico en el momento de la caída es:

<T_i, A, 10, 20>
<T_j, A, 10, 30>
<T_j comprometida>

Si se rehace primero, A tomará el valor 30; y luego, al deshacer, A acabará valiendo 10, lo cual es incorrecto. El valor final de A debe ser 30, lo que puede garantizarse si se deshace antes de rehacer.

17.7. GESTIÓN DE LA MEMORIA INTERMEDIA

En este apartado se consideran varios detalles sutiles que son esenciales para la implementación de un esquema de recuperación que garantice la consistencia de los datos y que lleve asociado una sobrecarga mínima respecto a la interacción con la base de datos.

17.7.1. Registro histórico con memoria intermedia

Anteriormente se supuso que se escribe cada registro del registro histórico en almacenamiento estable en el mismo momento de su creación. Esta suposición impone una sobrecarga muy alta en la ejecución del sistema por las siguientes razones. Habitualmente, la unidad de escritura en almacenamiento estable es el bloque. En la mayoría de los casos un registro del registro histórico es mucho más pequeño que un bloque. Así, la escritura de cada registro del registro histórico se traduce en una escritura mucho mayor en el nivel físico. Además de esto, como se vio en el Apartado 17.2.2, la escritura de un bloque en almacenamiento estable puede involucrar varias operaciones de escritura en el nivel físico.

El coste de realizar la escritura en almacenamiento estable de un bloque es suficientemente elevado para que sea deseable escribir de una sola vez varios registros del registro histórico. Para hacer esto se escriben los registros del registro histórico en una memoria intermedia almacenada en la memoria principal en la que perma-

necen durante un tiempo hasta que se guardan en almacenamiento estable. Se pueden acumular varios registros del registro histórico en la memoria intermedia del registro histórico y escribirse en almacenamiento estable con una sola operación. El orden de los registros del registro histórico en el almacenamiento estable debe ser exactamente el mismo orden en el que fueron escritos en la memoria intermedia del registro histórico.

Debido a la utilización de la memoria intermedia en el registro histórico, antes de ser escrito en almacenamiento estable, un registro del registro histórico puede permanecer únicamente en memoria principal (almacenamiento volátil) durante un espacio de tiempo considerable. Como esos registros se perderían si el sistema cayese, es necesaria la imposición de nuevos requisitos sobre las técnicas de recuperación para garantizar la atomicidad de las transacciones:

- Después de que el registro <T_j comprometida> se haya escrito en almacenamiento estable, la transacción T_j pasa al estado comprometida.
- Antes de escribir en almacenamiento estable el registro <T_j comprometida>, todos los registros del registro histórico pertenecientes a la transacción T_j se deben escribir en almacenamiento estable.
- Antes de que un bloque de datos en memoria principal se pueda escribir en la base de datos (en almacenamiento no volátil) todos los registros del registro histórico pertenecientes a los datos de ese bloque se deben escribir en almacenamiento estable.

  Esta última ligadura se denomina regla de **registro de escritura anticipada (REA)**. (Estrictamente, la regla REA sólo necesita que haya sido puesta en almacenamiento estable la información concerniente a la operación deshacer y permite que la información relativa a la operación rehacer pueda escribirse más tarde. La diferencia es relevante en aquellos sistemas en los que la información para rehacer y deshacer se guarda en registros del registro histórico independientes.)

Las reglas anteriores representan situaciones en las que ciertos registros del registro histórico *deben* haber sido escritos en almacenamiento estable. No se produce ningún problema como resultado de la escritura de los registros del registro histórico *antes* de que sea necesaria. Así, cuando el sistema decide que es necesario escribir en almacenamiento estable un registro del registro histórico, puede escribir un bloque entero de ellos si hay suficientes registros en memoria principal como para llenar un bloque. Si no hay suficientes registros para llenar el bloque, se forma un bloque parcialmente lleno con todos los registros que hubiera en memoria principal y se ponen en almacenamiento estable.

La escritura en disco de la memoria intermedia del registro histórico se denomina a veces **forzar el registro histórico**.

17.7.2. Base de datos con memoria intermedia

En el Apartado 17.2 se describió el uso de una jerarquía de almacenamiento de dos niveles. La base de datos se almacena en almacenamiento no volátil (disco) y, cuando es necesario, se traen a memoria principal los bloques de datos que hagan falta. Como la memoria principal suele ser mucho más pequeña que la base de datos completa, puede ser necesaria la sobreescritura de un bloque B_1 en memoria principal cuando es necesario traer a memoria otro bloque B_2. Si B_1 ha sido modificado, B_1 se debe escribir antes de traer B_2. Como se discutió en el Apartado 11.5.1, en el Capítulo 11, esta jerarquía de almacenamiento se corresponde con el concepto usual de *memoria virtual*.

Las reglas que rigen la escritura de registros del registro histórico limitan la libertad del sistema para escribir bloques de datos. Si la lectura del bloque B_2 provoca que el bloque B_1 tenga que escribirse en almacenamiento estable, todos los registros del registro histórico pertenecientes a los datos de B_1 se deben escribir en almacenamiento estable antes de la escritura de B_1. Por tanto, la secuencia de acciones que ha de llevar a cabo el sistema sería ésta:

- Escritura en almacenamiento estable de los registros del registro histórico hasta que todos los registros pertenecientes al bloque B_1 se hayan escrito.
- Escritura en disco del bloque B_1.
- Lectura del bloque B_2 desde el disco a la memoria principal.

Es un hecho importante el que no se produzcan escrituras sobre el bloque B_1 mientras que se lleva a cabo la anterior secuencia de acciones. Esto puede garantizarse, como se explica a continuación, utilizando un medio de bloqueo especial. Antes de que una transacción realice una escritura sobre un elemento de datos debe adquirir un bloqueo en exclusiva sobre el bloque en el que reside el citado elemento de datos. Inmediatamente después de haber realizado la modificación, el bloqueo se puede liberar. Antes de la escritura de un bloque, el sistema obtiene un bloqueo exclusivo sobre ese bloque para asegurarse de que ninguna transacción está modificándolo. Una vez que la escritura del bloque se ha completado, el bloqueo se libera. Los bloqueos de corta duración se denominan con frecuencia **pestillos**. Los pestillos y los bloqueos utilizados por el sistema de control de concurrencia se tratan de forma diferente. Como resultado, los pestillos pueden liberarse sin necesidad de cumplir ningún protocolo de bloqueo, como el bloqueo de dos fases requerido por el sistema de control de concurrencia.

Para ilustrar la necesidad del requisito de registro histórico con escritura anticipada considérese el ejemplo bancario con las transacciones T_0 y T_1. Supóngase que el estado del registro histórico es

<T_0 iniciada>
<T_0, A, 1.000, 950>

y que la transacción T_0 realiza una operación leer(B). Supóngase también que el bloque en el que se encuentra B no está en memoria principal y que esa memoria principal está llena. Supóngase por último que el bloque en el que reside A es el elegido para la sustitución y, por tanto, ha de escribirse en disco. Si el sistema escribe este bloque en disco y luego sucede una caída, los valores para las cuentas A, B y C en la base de datos son 950, 2.000 y 700 € respectivamente. Este estado de la base de datos es inconsistente. Sin embargo, según los requisitos de la regla REA, el registro del registro histórico

$$<T_0, A, 1.000, 950>$$

debe escribirse en almacenamiento estable antes de producirse la escritura del bloque en el que se encuentra A. El sistema puede usar ese registro del registro histórico durante la recuperación para devolver a la base de datos a un estado consistente.

17.7.3. El papel del sistema operativo en la gestión de la memoria intermedia

La memoria intermedia de la base de datos puede gestionarse usando uno de estos dos enfoques:

1. El sistema de base de datos reserva parte de la memoria principal para utilizarla como memoria intermedia y es él, en vez del sistema operativo, el que se encarga de gestionarlo. El sistema de base de datos gestiona la transferencia de los bloques de datos de acuerdo con los requisitos del Apartado 17.7.2.

   El inconveniente de este enfoque es que limita la flexibilidad en la utilización de la memoria principal. El tamaño de la memoria intermedia no debe ser muy grande para que otras aplicaciones tengan suficiente espacio disponible en la memoria principal para sus propias necesidades. Sin embargo, incluso cuando ninguna otra aplicación esté en ejecución, la base de datos no podrá hacer uso de toda la memoria disponible. Asimismo, aquellas aplicaciones que no tienen nada que ver con la base de datos no pueden usar la región de la memoria reservada para la memoria intermedia de la base de datos aunque no se estén utilizando algunas de las páginas almacenadas en la memoria intermedia.

2. El sistema de base de datos implementa su memoria intermedia dentro de la memoria virtual del sistema operativo. Como el sistema operativo conoce las necesidades de memoria de todos los procesos del sistema, es lógico que pueda decidir los bloques de la memoria intermedia que deben escribirse al disco y el momento en el que debe realizarse esta escritura. Pero para garantizar los requisitos del registro histórico de escritura anticipada del Apartado 17.7.1, el sistema operativo no debería realizar él mismo la escritura de las páginas de la base de datos de la memoria intermedia, sino que debería pedírselo al sistema de base de datos para que fuera éste quien forzara la escritura de los bloques de la memoria intermedia. El sistema de base de datos, después de escribir en almacenamiento estable los registros del registro histórico relevantes, forzaría la escritura en la base de datos de los bloques de la memoria intermedia.

   Por desgracia, casi todos los sistemas operativos de la generación actual ejercen un control completo sobre la memoria virtual. El sistema operativo reserva espacio en el disco para almacenar las páginas de memoria virtual que no se encuentran en ese momento en la memoria principal; este espacio se denomina **espacio de intercambio**. Si el sistema operativo decide escribir un bloque B_x, ese bloque se escribe en el espacio de intercambio del disco por lo que el sistema de base de datos no tiene forma de controlar la escritura de los bloques de la memoria intermedia.

   Por consiguiente, si la memoria intermedia de la base de datos está en la memoria virtual, las transferencias entre los archivos de la base de datos y la memoria intermedia en memoria virtual deben estar gestionadas por el sistema de base de datos, hecho que subraya el cumplimiento de los requisitos del registro histórico de escritura anticipada que se vieron.

   Este enfoque puede provocar una escritura adicional de datos en el disco. Si el sistema operativo realiza la escritura de un bloque B_x, éste no se escribe en la base de datos sino que se escribe en el espacio de intercambio que utiliza la memoria virtual del sistema operativo. Cuando la base de datos necesita escribir B_x, el sistema operativo puede necesitar primero leer B_x de su espacio de intercambio. Así, en lugar de realizar una sola escritura de B_x, son necesarias dos escrituras (una del sistema operativo y otra del sistema de base de datos) y una lectura adicional.

Aunque ambos enfoques tienen algunos inconvenientes, debe elegirse cualquiera de los dos excepto si el sistema operativo está diseñado para soportar los requisitos del registro histórico de base de datos. De los sistemas operativos actuales sólo unos pocos, como el sistema operativo Mach, soportan estos requisitos.

17.8. FALLO CON PÉRDIDA DE ALMACENAMIENTO NO VOLÁTIL

Hasta ahora sólo se ha considerado el caso en el que un fallo conduce a la pérdida de información residente en almacenamiento volátil mientras que el contenido del almacenamiento no volátil permanecía intacto. A pesar de que es raro encontrarse con un fallo en el que se pierda información de almacenamiento no volátil, es necesario prepararse para afrontar este tipo de fallos. En este apartado se hablará sólo del almacenamiento en disco. La argumentación puede aplicarse también a otras clases de almacenamiento no volátil.

La idea básica es **volcar** periódicamente (una vez al día) el contenido entero de la base de datos en almacenamiento estable. Por ejemplo, puede volcarse la base de datos en una o más cintas magnéticas. Se utilizará el volcado más reciente para que la base de datos recupere un estado consistente cuando ocurra un fallo que conduzca a la pérdida de algunos bloques físicos de la base de datos. Una vez que se complete esta operación, el sistema utilizará el registro histórico para llevar al sistema de base de datos al último estado consistente en el que estuvo antes de producirse el fallo.

Más precisamente, ninguna transacción puede estar activa durante el procedimiento de volcado y tendrá lugar una secuencia de acciones similar a la utilizada en los puntos de revisión:

1. Escribir en almacenamiento estable todos los registros del registro histórico que residan en ese momento en memoria principal.
2. Escribir en disco todos los bloques de la memoria intermedia.
3. Copiar el contenido de la base de datos en almacenamiento estable.
4. Escribir el registro del registro histórico <revisión> en almacenamiento estable.

Los pasos 1, 2 y 4 se corresponden con los tres pasos utilizados para realizar un punto de revisión en el Apartado 17.4.3.

Para la recuperación por pérdida de almacenamiento no volátil se restituye la base de datos en el disco utilizando el último volcado realizado. Entonces se consulta el registro histórico y se rehacen todas las transacciones que se hubieran comprometido desde que se efectuó ese último volcado. Nótese que no es necesario ejecutar ninguna operación deshacer.

Un volcado del contenido de una base de datos se denomina también **volcado de archivo** ya que pueden archivarse los volcados y utilizarlos más tarde para examinar estados anteriores de la base de datos. Los volcados de una base de datos y la realización de puntos de revisión de las memorias intermedias son dos procesos análogos.

El procedimiento de volcado simple que se ha descrito anteriormente es costoso debido a las dos razones siguientes. En primer lugar, debe copiarse en almacenamiento estable la base de datos entera, lo que conlleva una considerable transferencia de datos. En segundo lugar, se pierden ciclos de UCP porque se detiene el procesamiento de transacciones durante el procedimiento de volcado. Se han desarrollado esquemas de **volcado difuso** que permiten que las transacciones sigan activas mientras se realiza el volcado. Son esquemas similares a los de los puntos de revisión difusos. Para obtener más detalles véanse las notas bibliográficas.

17.9. TÉCNICAS AVANZADAS DE RECUPERACIÓN**

Las técnicas de recuperación descritas en el Apartado 17.6 requieren que, una vez que una transacción modifica un elemento de datos, ninguna otra pueda modificar el mismo elemento de datos hasta que la primera se comprometa o se retroceda. Utilizando el bloqueo estricto de dos fases se garantiza esa condición. Aunque el bloqueo estricto de dos fases es aceptable para los registros en las relaciones, como se vio en el Apartado 16.9, provoca un decremento significativo en la concurrencia cuando se aplica sobre determinadas estructuras, como las páginas indexadas con árboles B^+.

Para incrementar la concurrencia puede usarse el algoritmo de control de concurrencia en árboles B^+ descrito en el Apartado 16.9 y así permitir que los bloqueos se liberen rápidamente, no con el procedimiento de dos fases. Como consecuencia, sin embargo, no serán aplicables las técnicas de recuperación del Apartado 17.6. Se han propuesto varias técnicas de recuperación alternativas que pueden aplicarse incluso con bloqueos de liberación rápida. Estos esquemas se pueden usar en varias aplicaciones, no sólo para recuperar árboles B^+. En primer lugar se describe un esquema de recuperación avanzado que soporta los bloqueos de liberación rápida. A continuación se describe el esquema de recuperación ARIES, que se usa ampliamente en la industria. ARIES es más complejo que el esquema de recuperación avanzado descrito anteriormente, pero incorpora varias optimizaciones para minimizar el tiempo de recuperación, y proporciona varias características útiles.

17.9.1. Registro de deshacer lógico

En las operaciones donde los bloqueos son de liberación rápida no pueden realizarse las operaciones deshacer escribiendo simplemente el valor anterior de los elementos de datos. Considérese una transacción T que inserta una entrada en un árbol B^+ y, siguiendo el protocolo de control de concurrencia con árboles B^+, libera algunos bloqueos después de completarse la operación inserción, pero antes de que la transacción se comprometa. Después de liberar los bloqueos, otras transacciones pueden realizar inserciones o borrados posteriores cambiando de este modo las páginas del árbol B^+.

Incluso aunque la operación libere rápidamente algunos bloqueos debe conservar suficientes bloqueos como para garantizar que ninguna otra transacción tenga permiso para ejecutar cualquier operación conflictiva (como leer o borrar el valor insertado). Por este motivo, el protocolo de control de concurrencia con árboles B^+ que se estudió anteriormente mantiene ciertos bloqueos en las hojas del árbol B^+ hasta el final de la transacción.

Considérese ahora cómo realizar retrocesos de transacciones. Si se usa una operación **deshacer física**, es decir, si durante el retroceso se escriben los valores anteriores de los nodos internos del árbol B^+ (antes de ejecutar la operación inserción), podrían perderse algunas de las modificaciones realizadas por inserciones o borrados ejecutados posteriormente por otras transacciones. La operación inserción no debe deshacerse así, sino con una operación **deshacer lógica**, esto es, mediante la ejecución de una operación borrado en este caso.

Así, cuando finaliza la acción de inserción, antes de que libere ningún bloqueo, escribe en el registro histórico un registro <T_i, Oj, fin-operación, D>, donde D denota la información para deshacer y O_j es un identificador único de la operación. Por ejemplo, si la operación insertó una entrada en el árbol B^+, la información para deshacer indicaría lo que habría que borrar del árbol B^+. Este registro histórico de información acerca de las operaciones se denomina **registro histórico lógico**. En cambio, el registro histórico de la información sobre el valor anterior y el nuevo valor se denomina **registro histórico físico** y los correspondientes registros del registro histórico se llaman **registros del registro histórico físico**.

Las operaciones de inserción y borrado son ejemplos de un tipo de operaciones que requiere operaciones deshacer lógicas, ya que liberan rápidamente los bloqueos. Estas operaciones se denominan **operaciones lógicas**. Antes de que una operación lógica dé comienzo, escribe en el registro histórico un registro <T_i, O_j, inicio-operación> donde O_j es el identificador único de la operación. Durante la ejecución de la operación se registran todas las modificaciones realizadas por la operación de forma normal. De esta manera se escribe para cada modificación la información habitual sobre el valor anterior y el nuevo valor. Al finalizar la operación se escribe en el registro histórico un registro de fin-operación como se describió anteriormente.

17.9.2. Retroceso de transacciones

Considérese primero el retroceso de transacciones durante el modo de operación normal (esto es, no durante la fase de recuperación). Se recorre el registro histórico hacia atrás y se usan los registros del registro histórico pertenecientes a la transacción para devolver a los elementos de datos sus valores anteriores. A diferencia del retroceso durante una operación normal, se escriben registros especiales sólo para la operación rehacer de la forma <T_i, X_j, V> que contienen el valor V con el que se ha restaurado el elemento de datos X_j durante el retroceso. Estos registros del registro histórico se denominan a veces **registros de compensación del registro histórico**. Estos registros no necesitan información para deshacer puesto que nunca es necesario realizar esta operación.

Se toman acciones especiales cuando se encuentra un registro del registro histórico de la forma <T_i, O_j, fin-operación, D>:

1. Se retrocede la operación mediante la información para deshacer D que se encuentra en el registro del registro histórico. Las modificaciones realizadas durante el retroceso de la operación se registran de la misma manera que las modificaciones realizadas cuando la operación se ejecutó por primera vez. En otras palabras, el sistema registra información de deshacer física para las actualizaciones realizadas durante el retroceso, en lugar de usar registros de compensación del registro histórico. Esto es debido a que puede ocurrir una caída mientras la operación deshacer lógica se encuentre en curso, y el sistema debe completar durante la recuperación la operación deshacer lógica, usando la información deshacer física, y después realizar de nuevo la operación deshacer lógica, como se verá en el apartado 17.9.4.

   Al final de la operación de retroceso, en lugar de generar un registro del registro histórico <T_i, O_j, fin-operación, D>, el sistema genera <T_i, O_j, abortar-operación, D>.
2. Cuando continúa el recorrido hacia atrás del registro histórico se ignoran todos los registros del registro histórico de la transacción hasta que se encuentra el registro <T_i, O_j, inicio-operación>. Una vez que se encuentra en el registro histórico el registro inicio-operación se procesan de nuevo de manera normal los registros del registro histórico referentes a la transacción.

Obsérvese que la omisión de los registros del registro histórico físico cuando se encuentra el registro fin-ope-

ración durante el retroceso asegura que los valores anteriores del registro no se usen para el retroceso una vez que termine la operación.

Si el sistema encuentra un registro <T_i, O_j, abortar-operación, D>, omite todos los registros precedentes hasta que se encuentra el registro <T_i, O_j, inicio-operación>. Estos registros precedentes se deben omitir para evitar varios retrocesos de la misma operación en el caso de que haya una caída durante el retroceso anterior, y de que la transacción se haya retrocedido parcialmente. Cuando se haya retrocedido la transacción T_i, el sistema añade al registro <T_i abortar>.

Si sucede un fallo mientras se está ejecutando una operación lógica, el registro fin-operación de la operación no se encontrará cuando se retroceda la transacción. Sin embargo, para cada actualización realizada por la operación hay disponible en el registro histórico información para deshacer (como valor anterior en los registros del registro histórico físico). Los registros del registro histórico físico se usarán para retroceder la operación incompleta.

17.9.3. Puntos de revisión

Los puntos de revisión se llevan a cabo como se describió en el Apartado 17.6. Se suspenden temporalmente las modificaciones sobre la base de datos y se llevan a cabo las siguientes acciones:

1. Se escriben en almacenamiento estable todos los registros del registro histórico que se encuentren en ese momento en la memoria principal.
2. Se escriben en disco todos los bloques de la memoria intermedia que se hayan modificado.
3. Se escribe en almacenamiento estable el registro <revisión L>, donde L es una lista de todas las transacciones activas.

17.9.4. Recuperación al reiniciar

Las acciones de recuperación se realizan en dos fases cuando se vuelve a iniciar el sistema de base de datos después de un fallo:

1. En la **fase rehacer** se vuelven a realizar modificaciones de *todas* las transacciones mediante la exploración hacia delante del registro histórico a partir del último punto de revisión. Los registros del registro histórico que se vuelven a ejecutar incluyen los registros de transacciones que se retrocedieron antes de la caída del sistema y de las que no se habían comprometido cuando ocurrió la caída. Los registros del registro histórico son los registros habituales de la forma <T_i, X_j, V_1, V_2>, así como los registros especiales del registro histórico de la forma <T_i, X_j, V_2>; el elemento de datos X_j adquiere el valor V_2 en cualquier caso. Esta fase también determina todas las transacciones que o bien se encuentran en la lista de transacciones del registro del punto de revisión, o bien comenzaron más tarde, pero no tienen ni el registro <T_i abortada> ni el registro <T_i comprometida> en el registro histórico. Todas estas transacciones deben retrocederse y sus identificadores de transacción se ponen en una *lista-deshacer*.
2. En la **fase deshacer** se retroceden todas las transacciones de la *lista-deshacer*. Se realiza el retroceso recorriendo el registro histórico hacia atrás empezando por el final. Cuando se encuentra un registro del registro histórico perteneciente a una transacción de la *lista-deshacer* se realizan las operaciones para deshacer de la misma manera que si el registro se hubiera encontrado durante el retroceso de una transacción fallida. Así pues, se ignoran los registros del registro histórico de una transacción que preceden a un registro fin-operación, pero que se encuentran detrás del correspondiente registro inicio-operación.

   Cuando se encuentra en el registro histórico un registro <T_i iniciada> para una transacción T_i de la *lista-deshacer*, se escribe en el registro histórico un registro <T_i abortada>. El recorrido hacia atrás del registro histórico finaliza cuando se encuentran los registros <T_i iniciada> para todas las transacciones de la *lista-deshacer*.

La fase rehacer de la recuperación al reiniciar reproduce cada registro físico del registro histórico desde que tuvo lugar el último punto de revisión. En otras palabras, esta fase de la recuperación al reiniciar repite todas las acciones de modificación que fueron ejecutadas después del punto de revisión y cuyos registros alcanzaron un registro histórico estable. Se incluyen aquí las acciones de transacciones incompletas y las acciones llevadas a cabo para retroceder transacciones fallidas. Se repiten las acciones en el mismo orden en el que se llevaron a cabo; de aquí que este proceso se denomine **repetición de la historia**. Al repetir la historia se simplifican bastante los esquemas de recuperación.

Nótese que si una operación deshacer estaba en curso cuando ocurrió la caída del sistema, se encontrarían los registros del registro histórico físico escritos durante la operación deshacer, y la operación deshacer parcial se desharía según estos registros del registro histórico físico. Después de ello, el registro fin-operación de la operación original se encontraría durante la recuperación, y la operación deshacer se ejecutaría de nuevo.

17.9.5. Revisión difusa

La revisión difusa descrita en el Apartado 17.6.3 requiere que, mientras se efectúa el punto de revisión, se suspendan temporalmente todas las modificaciones de la base de datos. Si el número de páginas de la memoria

intermedia es grande, un punto de revisión puede llevar mucho tiempo, lo que puede resultar en una interrupción inaceptable en el procesamiento de transacciones.

Para evitar estas interrupciones es posible modificar la técnica para permitir modificaciones después de haber escrito en el registro histórico el registro revisión, pero antes de escribir en disco los bloques de la memoria intermedia que han sufrido modificaciones. El punto de revisión así generado recibe el nombre de **punto de revisión difuso**.

Dado que las páginas se escriben en disco sólo después de que se haya escrito el registro revisión, es posible que el sistema caiga antes de que todas las páginas se hayan escrito. Por tanto, los puntos de revisión en disco pueden estar incompletos. Una forma de manejar los puntos de revisión incompletos es la siguiente. La ubicación del registro de revisión en el registro histórico del último punto de revisión completado se almacena en una posición fijada, el último punto de revisión, en disco. El sistema no actualiza esta información cuando escribe el registro revisión. En su lugar, antes de escribirlo, crea una lista de todos los bloques de memoria intermedia modificados. La información del último punto de revisión se actualiza sólo cuando todos los bloques de memoria intermedia de la lista de bloques modificados se hayan escrito en disco.

Incluso con la revisión difusa, un bloque de memoria intermedia no se debe actualizar mientras se esté escribiendo en disco, aunque otros bloques sí se pueden actualizar concurrentemente. Una vez que todos los bloques han sido escritos en disco debe seguirse el protocolo de registro histórico de escritura anticipada.

Nótese que en este esquema sólo se utiliza el proceso de registro histórico lógico para deshacer mientras que el proceso de registro histórico físico se usa tanto para deshacer como para rehacer. Existen otros esquemas de recuperación que utilizan el proceso de registro histórico lógico para rehacer. Para deshacer lógicamente, el estado de la base de datos en disco debe ser **consistente en cuanto a operaciones**, es decir, no debería tener efectos parciales de ninguna operación. Es difícil garantizar esta consistencia de la base de datos en disco si una operación puede afectar a más de una página, puesto que no es posible escribir más de una página de forma atómica. Por tanto, el registro histórico deshacer lógico se restringe usualmente sólo a operaciones que afectan a una única página; se verá la forma de tratar estas operaciones rehacer lógicas en el Apartado 17.9.6. En cambio, las operaciones deshacer lógicas se realizan sobre un estado consistente en cuanto a operaciones de la base de datos repitiendo la historia y después realizando la operación deshacer física de las operaciones completadas parcialmente.

17.9.6. ARIES

El método de recuperación ARIES representa a los métodos actuales de recuperación. La técnica de recuperación avanzada que se ha descrito se ha modelado después de ARIES, pero se ha simplificado significativamente para ilustrar los conceptos clave y hacerlo más fácil de comprender. En cambio, ARIES utiliza varias técnicas para reducir el tiempo de recuperación y para reducir la sobrecarga de los puntos de revisión. En particular, ARIES es capaz de evitar rehacer muchas operaciones registradas que ya se han realizado y de reducir la cantidad de información registrada. El precio pagado es una mayor complejidad, pero los beneficios merecen la pena.

Las diferencias principales entre ARIES y el algoritmo de recuperación avanzada expuesto son que ARIES:

1. Usa un **número de secuencia del registro histórico** para identificar a los registros del registro histórico, y en el uso de estos números en las páginas de la base de datos para identificar las operaciones que se han realizado sobre una página de la base de datos.
2. Soporta operaciones **rehacer fisiológicas**, que son físicas en el sentido en que la página afectada está físicamente identificada, pero que pueden ser lógicas en la página.

   Por ejemplo, el borrado de un registro de una página puede resultar en que muchos otros registros de la página se desplacen si se usa una estructura de páginas con ranuras. Con el registro histórico rehacer físico, todos los bytes de la página afectada por el desplazamiento de los registros se deben registrar. Con el registro histórico fisiológico, la operación borrado se puede registrar, resultando en un registro mucho más pequeño. Al rehacer la operación borrado se borraría el registro y se desplazarían los registros que fuese necesario.
3. Usa una **tabla de páginas desfasadas** para minimizar las operaciones rehacer innecesarias durante la recuperación. Las páginas desfasadas son las que se han actualizado en memoria pero su versión en disco no.
4. Usa un esquema de revisión difusa que sólo registra información sobre las páginas desfasadas e información asociada, y no requiere siquiera la escritura de las páginas desfasadas a disco. Procesa las páginas desfasadas en segundo plano continuamente, en lugar de escribirlas durante los puntos de revisión.

En el resto de este apartado se proporciona una visión general de ARIES. Las notas bibliográficas listan referencias que proporcionan una descripción completa de ARIES.

17.9.6.1. Estructuras de datos

Cada registro del registro histórico de ARIES tiene un **número de secuencia del registro histórico (NSR)** que

lo identifica unívocamente. El número es conceptualmente tan sólo un identificador lógico cuyo valor es mayor para los registros que aparecen después en el registro histórico. En la práctica, el NSR se genera de forma que también se puede usar para localizar el registro del registro histórico en disco. Normalmente, ARIES divide el registro histórico en varios archivos de registro histórico, cada uno con un número de archivo. Cuando un archivo crece hasta un determinado límite, ARIES añade los nuevos registros del registro histórico en un nuevo archivo; el nuevo archivo de registro histórico tiene un número de archivo que es 1 mayor que el anterior archivo. El NSR consiste en un número de archivo y un desplazamiento dentro del archivo.

Cada página también mantiene un identificador denominado **NSRPágina**. Cada vez que se aplica una operación (física o lógica) en la página, la operación almacena el NSR de su registro en el campo NSRPágina de la página. Durante la fase rehacer de la recuperación cualquier registro con un NSR menor o igual que el NSRPágina de la página no se debería ejecutar, ya que sus acciones ya están reflejadas en la página. En combinación con un esquema para el registro de los NSRPágina como parte de los puntos de revisión, que se presenta más adelante, ARIES puede evitar incluso leer muchas páginas cuyas operaciones registradas ya se han reflejado en el disco. Por tanto, el tiempo de recuperación se reduce significativamente.

El NSRPágina es esencial para asegurar la idempotencia en presencia de operaciones rehacer fisiológicas, ya que volver a aplicar una operación rehacer fisiológica que ya se haya aplicado a una página podría causar cambios incorrectos en una página.

Las páginas no se deberían enviar a disco mientras se esté realizando una actualización, dado que las operaciones fisiológicas no se pueden rehacer sobre el estado parcialmente actualizado de la página en disco. Por tanto, ARIES usa pestillos sobre las páginas de la memoria intermedia para evitar que se escriban en disco mientras se actualicen. Los pestillos de las páginas de la memoria intermedia sólo se liberan cuando se completan las actualizaciones, y el registro del registro histórico para la actualización se haya escrito en el registro histórico.

Cada registro del registro histórico también contiene el NSR del registro anterior de la misma transacción. Este valor, almacenado en el campo NSRAnterior, permite que se encuentren los registros del registro histórico anteriores sin necesidad de leer el registro histórico completo. En ARIES hay registros especiales *sólo-rehacer* generados durante el retroceso de transacciones, denominados **registros de compensación del registro histórico (RCR)**. Sirven para el mismo propósito que los registros sólo-rehacer del registro histórico del esquema de recuperación avanzado. Además juegan el papel de los registros abortar-operación de ese esquema. Los RCR tienen un campo extra denominado DeshacerSiguienteNSR, que registra el NSR del registro que hay que deshacer a continuación cuando se retrocede la transacción. Este campo sirve para el mismo propósito que el identificador de operaciones en el registro abortar-operación del esquema anterior, que ayuda a omitir los registros que ya se hayan retrocedido. La **TablaPáginasDesfasadas** contiene una lista de páginas que se han actualizado en la memoria intermedia de la base de datos. Para cada página se almacena el NSRPágina y un campo denominado RegNSR que ayuda a identificar los registros que ya se han aplicado a la versión en disco de la página. Cuando se inserta una página en la TablaPáginasDesfasadas (cuando se modifica por primera vez en el grupo de memorias intermedias) el valor de RegNSR se establece en el fin actual del registro histórico. Cada vez que se envía una página a disco, la página se elimina de la TablaPáginasDesfasadas.

El **registro punto de revisión del registro histórico** contiene la TablaPáginasDesfasadas y una lista de transacciones activas. Para cada transacción, el registro punto de revisión del registro histórico también anota ÚltimoNSR, el NSR del último registro escrito por la transacción. Una posición fijada en disco también anota el NSR del último registro punto de revisión del registro histórico (completado).

17.9.6.2. Algoritmo de recuperación

ARIES recupera de una caída del sistema en tres fases:

- **Paso de análisis.** Este paso determina las transacciones que hay que deshacer, las páginas que están desfasadas en el momento de la caída y el NSR en el que debería comenzar el paso rehacer.
- **Paso rehacer.** Este paso comienza en una posición determinada durante el análisis y realiza una operación rehacer, repitiendo la historia, para llevar a la base de datos al estado anterior a la caída.
- **Paso deshacer.** Este paso retrocede todas las transacciones incompletas en el momento de la caída.

Paso de análisis. El paso de análisis busca el último registro punto de revisión del registro histórico completado y lee la TablaPáginasDesfasadas en este registro. A continuación establece RehacerNSR al mínimo RegistroNSR de las páginas de TablaPáginasDesfasadas. Si no hay páginas desfasadas, establece RehacerNSR al NSR del registro punto de revisión del registro histórico. El paso rehacer comienza explorando el registro histórico desde RehacerNSR. Todos los registros anteriores a este punto ya se han aplicado a las páginas de la base de datos en el disco. El paso de análisis establece inicialmente la lista de transacciones que se deben deshacer, *lista-deshacer*, a la lista de transacciones en el registro punto de revisión del registro histórico. El paso de análisis también lee del registro punto de revisión del registro histórico los NSR del último registro del registro histórico de cada transacción de las *lista-deshacer*.

El paso de análisis continúa examinando hacia delante desde el punto de revisión. Cada vez que encuentra un registro de una transacción que no esté en la *lista-deshacer*, añade la transacción a la *lista-deshacer.* Cada vez que encuentra un registro de fin de transacción, borra la transacción de la *lista-deshacer*. Todas las transacciones que queden en la *lista-deshacer* al final del análisis se deben retroceder más tarde en el paso deshacer. El paso de análisis también almacena el último registro de cada transacción en la *lista-deshacer*, que se usa en el paso deshacer.

El paso de análisis también actualiza TablaPáginasDesfasadas cada vez que encuentra un registro del registro histórico de la actualización de una página. Si la página no está en la TablaPáginasDesfasadas, el paso de análisis la añade a ella y establece el RegistroNSR de la página al NSR del registro.

Paso rehacer. El paso rehacer repite la historia volviendo a ejecutar cada acción sobre una página que no se haya reflejado en disco. El paso rehacer examina el registro histórico hacia delante a partir de RehacerNSR. Cada vez que encuentra un registro actualizar realiza:

1. Si la página no está en la TablaPáginasDesfasadas o el NSR del registro actualizar es menor que el RegistroNSR de la página de TablaPáginasDesfasadas, entonces el paso rehacer omite el registro.
2. En caso contrario, el paso rehacer extrae la página de disco y, si NSRPágina es menor que el NSR del registro, se rehace el registro.

Nótese que si cualquiera de las comprobaciones son negativas, entonces los efectos del registro del registro histórico ya han aparecido en la página. Si la primera comprobación es negativa, ni siquiera es necesario extraer la página de disco.

Paso deshacer y retroceso de transacciones. El paso deshacer es relativamente simple. Realiza una exploración hacia atrás del registro histórico, deshaciendo todas las transacciones de la *lista-deshacer.* Si se encuentra un RCR, usa el campo DeshacerSiguienteNSR para omitir los registros que ya se hayan retrocedido. En caso contrario, usa el campo NSRAnterior del registro para encontrar el siguiente a deshacer.

Cada vez que se usa un registro del registro histórico para realizar una operación deshacer (para el retroceso de transacciones durante el procesamiento normal del retroceso o durante el reinicio del paso deshacer) el paso deshacer genera un RCR conteniendo la acción deshacer realizada (que debe ser fisiológica). Establece DeshacerSiguienteNSR del RCR al valor NSRAnterior del registro actualizar del registro histórico.

17.9.6.3. Otras características

Entre otras de las características que proporciona ARIES se encuentran:

- **Independencia de recuperación.** Algunas páginas se pueden recuperar independientemente de otras, de forma que se pueden usar incluso cuando se estén recuperando otras. Si fallan algunas páginas del disco se pueden recuperar sin parar el procesamiento de transacciones en otras páginas.
- **Puntos de almacenamiento.** Las transacciones pueden registrar puntos de almacenamiento y se pueden retroceder parcialmente hasta un punto de almacenamiento. Esto puede ser muy útil en el manejo de interbloqueos, dado que las transacciones se pueden retroceder hasta un punto que permita la liberación de los bloqueos requeridos y luego reiniciarse desde ese punto.
- **Bloqueo de grano fino.** El algoritmo de recuperación ARIES se puede usar con algoritmos de control de concurrencia de índices que permiten el bloqueo en el nivel de tuplas de los índices, en lugar del bloqueo en el nivel de las páginas, lo que aumenta significativamente la concurrencia.
- **Optimizaciones de recuperación.** La TablaPáginasDesfasadas se puede usar para preextraer páginas durante la operación rehacer, en lugar de extraer una página sólo cuando el sistema encuentra un registro del registro histórico a aplicar a la página. La operación rehacer no válidos también es posible. Esta operación se puede posponer sobre una página que se vaya a extraer del disco y realizarse cuando se extraiga. Mientras tanto se pueden procesar otros registros.

En resumen, el algoritmo ARIES es un algoritmo de recuperación actual que incorpora varias optimizaciones diseñadas para mejorar la concurrencia, reducir la sobrecarga por el registro histórico y reducir el tiempo de recuperación.

17.10. SISTEMAS REMOTOS DE COPIAS DE SEGURIDAD

Los sistemas tradicionales de procesamiento de transacciones son sistemas centralizados o sistemas cliente-servidor. Esos sistemas son vulnerables frente a desastres ambientales como el fuego, las inundaciones o los terremotos. Hay una necesidad creciente de sistemas de procesamiento de transacciones que ofrezcan una disponibilidad elevada y que puedan funcionar pese a los desastres ambientales. Estos sistemas deben proporcio-

nar una **disponibilidad elevada**, es decir, el tiempo en que el sistema no es utilizable debe ser extremadamente pequeño.

Se puede obtener una disponibilidad elevada realizando el procesamiento de transacciones en un solo sitio, denominado **sitio principal**, pero tener un **sitio remoto copia de seguridad**, en el que se repliquen todos los datos del sitio principal. El sitio remoto copia de seguridad se denomina también a veces **sitio secundario**. El sitio remoto debe mantenerse sincronizado con el sitio principal, ya que las actualizaciones se realizan en el sitio principal. La sincronización se obtiene enviando todos los registros del registro histórico desde el sitio principal al sitio remoto copia de seguridad. El sitio remoto copia de seguridad debe hallarse físicamente separado del principal —por ejemplo, se puede ubicar en otra provincia— para que una catástrofe en el sitio principal no afecte al sitio remoto copia de seguridad. En la Figura 17.10 se muestra la arquitectura de los sistemas remotos para copias de seguridad.

Cuando falla el sitio principal, el sitio remoto copia de seguridad asume el procesamiento. En primer lugar, sin embargo, lleva a cabo la recuperación utilizando su copia (tal vez anticuada) de los datos del sitio principal y los registros del registro histórico recibidos del mismo. En realidad, el sitio remoto copia de seguridad lleva a cabo acciones de recuperación que se hubieran llevado a cabo en el sitio principal cuando éste se hubiera recuperado. Se pueden utilizar los algoritmos estándar de recuperación para la recuperación en el sitio remoto copia de seguridad con pocas modificaciones. Una vez realizada la recuperación, el sitio remoto copia de seguridad comienza a procesar transacciones.

La disponibilidad aumenta mucho en comparación con los sistemas con un solo sitio, dado que el sistema puede recuperarse aunque se pierdan todos los datos del sitio principal. El rendimiento de los sistemas remotos para copias de seguridad es mejor que el de los sistemas distribuidos con compromiso de dos fases.

Varios aspectos que deben abordarse al diseñar sistemas remotos para copias de seguridad son los siguientes:

- **Detección de fallos.** Al igual que en los protocolos para el manejo de fallos en sistemas distribuidos, es importante que el sistema remoto de copia de seguridad detecte que el sitio principal ha fallado. El fallo de las líneas de comunicación puede hacer creer al sitio remoto copia de seguridad que el sitio principal ha fallado. Para evitar este problema hay que mantener varios enlaces de comunicaciones con modos de fallo independientes entre el sitio principal y el sitio remoto copia de seguridad. Por ejemplo, además de la conexión de red puede haber otra conexión mediante módem por línea telefónica con servicio suministrado por diferentes compañías de telecomunicaciones. Estas conexiones pueden complementarse con la intervención manual de operadores, que se pueden comunicar por sistema telefónico.
- **Transferencia del control.** Cuando el sitio principal falla, el sitio copia de seguridad asume el procesamiento y se transforma en el nuevo sitio principal. Cuando el sitio principal original se recupera puede desempeñar el papel de sitio remoto copia de seguridad o volver a asumir el papel de sitio principal. En cualquiera de los casos, el sitio principal antiguo debe recibir un registro histórico de actualizaciones realizado por el sitio copia de seguridad mientras el sitio principal antiguo estaba fuera de servicio.

  La manera más sencilla de transferir el control es que el sitio principal antiguo reciba el registro histórico de operaciones rehacer del sitio copia de seguridad antiguo y se ponga al día con las actualizaciones aplicándolas de manera local. El sitio principal antiguo puede entonces actuar como sitio remoto copia de seguridad. Si hay que devolver el control, el sitio remoto copia de seguridad antiguo puede simular que ha fallado, lo que da lugar a que el sitio principal antiguo asuma el control.
- **Tiempo de recuperación.** Si el registro histórico del sitio remoto copia de seguridad se hace grande, la recuperación puede tardar mucho. El sitio remoto copia de seguridad puede procesar de manera periódica los registros rehacer del registro histórico que haya recibido y realizar un punto de

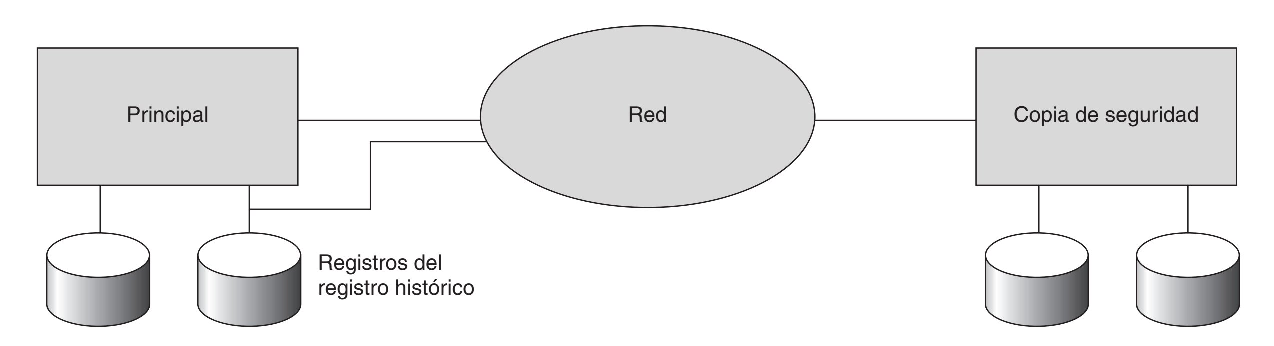


FIGURA 17.10. Arquitectura de los sistemas remotos de copias de seguridad.

revisión de manera que se puedan borrar las partes más antiguas del registro histórico. Como consecuencia se puede reducir el retraso antes de que el sitio remoto copia de seguridad asuma el control.

Una configuración de **relevo en caliente** puede hacer la toma del control por el sitio copia de seguridad casi instantáneo. En esta configuración el sitio remoto copia de seguridad procesa los registros rehacer del registro histórico según llegan, y aplica las actualizaciones de manera local. Tan pronto como se detecta el fallo del sitio principal, el sitio copia de seguridad completa la recuperación retrocediendo las transacciones incompletas y queda preparado para procesar las nuevas.

- **Tiempo de compromiso.** Para asegurar que las actualizaciones de una transacción comprometida sean duraderas no se debe declarar comprometida una transacción hasta que sus registros del registro histórico hayan alcanzado el sitio copia de seguridad. Este retraso puede dar lugar a una espera más prolongada para comprometer la transacción y, en consecuencia, algunos sistemas permiten grados inferiores de durabilidad. Los grados de durabilidad pueden clasificarse de la manera siguiente.
  - **Uno seguro.** Las transacciones se comprometen tan pronto como sus registros de compromiso del registro histórico se escriben en un almacenamiento estable en el sitio principal.

    El problema de este esquema es que puede que las actualizaciones de una transacción comprometida no hayan alcanzado el sitio copia de seguridad cuando éste asuma el control del procesamiento. Por tanto, puede parecer que las actualizaciones se han perdido. Cuando se recupere el sitio principal, las actualizaciones perdidas no se pueden mezclar directamente, dado que pueden entrar en conflicto con actualizaciones posteriores llevadas a cabo en el sitio copia de seguridad. Por tanto, puede que se necesite la intervención humana para devolver a la base de datos a un estado consistente.
  - **Dos muy seguro.** Las transacciones se comprometen tan pronto como sus registros de compromiso del registro histórico se escriben en un almacenamiento estable en el sitio principal y en el sitio copia de seguridad.

    El problema de este esquema es que el procesamiento de transacciones no puede continuar si alguno de los puntos no está operativo. Por tanto, la disponibilidad es realmente menor que en el caso de un solo sitio, aunque la posibilidad de la pérdida de datos es mucho más reducida.
  - **Dos seguro.** Este esquema es idéntico al esquema dos muy seguro si tanto el sitio principal como el sitio copia de seguridad están activos. Si sólo está activo el sitio principal se permite que la transacción se comprometa tan pronto como su registro de compromiso del registro histórico se escriba en un almacenamiento estable en el sitio principal.

    Este esquema proporciona mejor disponibilidad que el esquema dos muy seguro al tiempo que evita el problema de las transacciones perdidas afrontado por el esquema uno seguro. Da lugar a un compromiso más lento que el esquema uno seguro pero las ventajas generalmente superan los inconvenientes.

Varios sistemas comerciales de disco compartido proporcionan un nivel de tolerancia de fallos intermedio entre el de los sistemas centralizados y el de los sistemas remotos para copias de seguridad. En estos sistemas el fallo de una UCP no da lugar al fallo del sistema. En lugar de ello, otra UCP asume el control y lleva a cabo la recuperación. Las acciones de recuperación incluyen el retroceso de las transacciones que se ejecutaban en la UCP que falló y la recuperación de los bloqueos mantenidos por dichas transacciones. Dado que los datos se hallan en un disco compartido, no hace falta transferir registros del registro histórico. Sin embargo, se deberían proteger los datos contra el fallo del disco utilizando, por ejemplo, una organización de discos RAID.

Una forma alternativa de conseguir alta disponibilidad es usar una base de datos distribuida con los datos replicados en más de un sitio. Son necesarias transacciones para actualizar todas las réplicas de cualquier elemento de datos que actualicen. Las bases de datos distribuidas, incluyendo la réplica, se estudian en el Capítulo 19.

17.11. RESUMEN

- Un sistema informático, al igual que cualquier otro dispositivo eléctrico o mecánico, está sujeto a fallos. Estos fallos se producen por diferentes motivos incluyendo fallos de disco, cortes de corriente o fallos en el software. En cada uno de estos casos puede perderse información concerniente a la base de datos.
- Las transacciones pueden fallar, además de por un fallo del sistema, por otras razones como una violación de las restricciones de integridad o interbloqueos.
- El esquema de recuperación es una parte integrante del sistema de base de datos el cual es responsable de

la detección de fallos y del restablecimiento de un estado de la base datos anterior al momento de producirse el fallo.

- Los diferentes tipos de almacenamiento en una computadora son el volátil, el no volátil y el almacenamiento estable. Los datos del almacenamiento volátil, como ocurre con los guardados en la memoria RAM, se pierden cuando la computadora cae. Los datos del almacenamiento no volátil, como los guardados en un disco, no se pierden cuando la computadora cae, pero pueden perderse ocasionalmente debido a fallos de disco. Los datos del almacenamiento estable nunca se pierden.
- El almacenamiento estable de acceso en tiempo real puede aproximarse con discos con imagen u otras formas de RAID que proporcionan almacenamiento redundante de datos. El almacenamiento estable, sin conexión o de archivo, puede consistir en una serie de copias en cinta de los datos guardadas en un lugar físicamente seguro.
- El estado del sistema de base de datos puede no volver a ser consistente en caso de ocurrir un fallo; esto es, puede no reflejar un estado del mundo potencialmente alcanzable por la base de datos. Para preservar la consistencia es necesario que cada transacción sea atómica. Garantizar la propiedad de atomicidad es responsabilidad del esquema de recuperación. Existen básicamente dos esquemas para garantizar la atomicidad: basados en registro histórico y paginación en la sombra.
- En los esquemas basados en registro histórico todas las modificaciones se escriben en el registro histórico, el cual debe estar guardado en almacenamiento estable.
  - — En el esquema de modificación diferida, durante la ejecución de una transacción, se difieren todas las operaciones escribir hasta que la transacción se compromete parcialmente, momento en el que se utiliza la información del registro histórico asociada con la transacción para ejecutar las escrituras diferidas.
  - — Con la técnica de modificación inmediata, todas las modificaciones se aplican directamente sobre la base de datos. Si ocurre una caída se utiliza la información del registro histórico para conducir a la base de datos a un estado estable previo.

  Puede usarse la técnica de los puntos de revisión para reducir la sobrecarga que conlleva la búsqueda en el registro histórico y rehacer las transacciones.
- En la paginación en la sombra, durante la vida de una transacción se mantienen dos tablas de páginas: la tabla actual de páginas y la tabla de páginas sombra. Ambas tablas son idénticas cuando la transacción comienza. La tabla de páginas sombra y las tablas a las que apunta no sufren ningún cambio mientras dura la transacción. Cuando la transacción se compromete parcialmente se desecha la tabla de páginas sombra y la tabla actual se convierte en la nueva tabla de páginas. Si la transacción aborta, simplemente se desecha la tabla actual de páginas.
- La técnica de la paginación en la sombra no puede aplicarse si se permite que varias transacciones se ejecuten concurrentemente, pero el esquema basado en registro histórico sí.

  No se permite que ninguna transacción pueda modificar un elemento de datos que ya se ha modificado por una transacción incompleta. Para garantizar esta condición puede utilizarse el bloqueo estricto de dos fases.
- El procesamiento de transacciones se basa en un modelo de almacenamiento en el que la memoria principal contiene una memoria intermedia para el registro histórico, una memoria intermedia para la base de datos y una memoria intermedia para el sistema. La memoria intermedia del sistema alberga páginas de código objeto del sistema y áreas de trabajo local de las transacciones.
- Una implementación eficiente de un esquema de recuperación de datos requiere que el número de escrituras en la base de datos y en almacenamiento estable sea mínimo. Los registros del registro histórico pueden guardarse inicialmente en la memoria intermedia del registro histórico en almacenamiento volátil, pero se deben copiar en almacenamiento estable cuando se da una de estas dos condiciones:
  - — Deben escribirse en almacenamiento estable todos los registros del registro histórico pertenecientes a la transacción T_i antes de que el registro <T_i comprometida> se pueda escribir en almacenamiento estable.
  - — Deben escribirse en almacenamiento estable todos los registros del registro histórico pertenecientes a los datos de un bloque antes de que ese bloque de datos se escriba desde la memoria principal a la base de datos (en almacenamiento no volátil).
- Para recuperarse de los fallos que resultan en la pérdida de almacenamiento no volátil debe realizarse un volcado periódicamente (por ejemplo, una vez al día) del contenido entero de la base de datos en almacenamiento estable. Se usará el último volcado para devolver a la base de datos a un estado consistente previo cuando ocurra un fallo que conduzca a la pérdida de algún bloque físico de la base de datos. Una vez realizada esta operación se utilizará el registro histórico para llevar a la base de datos al estado consistente más reciente.
- Se han desarrollado técnicas avanzadas de recuperación para soportar técnicas de bloqueo de alta concurrencia, como las utilizadas para el control de concurrencia con árboles B^+. Estas técnicas se basan en el registro deshacer lógico y siguen el principio de repetir la historia. En la recuperación de un fallo del

sistema se realiza una fase rehacer utilizando el registro histórico seguida de una fase deshacer sobre el registro histórico para retroceder las transacciones incompletas.

- El esquema de recuperación ARIES es un esquema actual que soporta varias características para proporcionar mayor concurrencia, reducir la sobrecarga del registro histórico y permitir operaciones deshacer lógicas. El esquema procesa páginas continuamente y no necesita procesar todas las páginas en el momento de un punto de revisión. Usa números de secuencia del registro histórico (NSR) para implementar varias optimizaciones que reducen el tiempo de recuperación.
- Los sistemas remotos de copia de seguridad proporcionan un alto nivel de disponibilidad, permitiendo que continúe el procesamiento de transacciones incluso si se destruye el sitio primario por fuego, inundación o terremoto.

TÉRMINOS DE REPASO

- Alta disponibilidad
- ARIES
  - Número de secuencia del registro histórico (NSR)
  - NSRPágina
  - Operación rehacer fisiológica
  - Registros de compensación del registro histórico (RCR)
  - TablaPáginasDesfasadas
  - Registro punto de revisión
- Bloques
  - Bloques físicos
  - Bloques de memoria intermedia
- Clasificación de los fallos
  - Fallo de transacción
  - Error lógico
  - Error del sistema
  - Caída del sistema
  - Fallo de transferencia de datos
- Configuración de relevo en caliente
- Detección de fallos
- Escritura forzada
- Esquema de recuperación
- Fallo de disco
- Forzar el registro histórico
- Gestión de la memoria intermedia
- Idempotente
- Memoria intermedia de la base de datos
- Memoria intermedia de disco
- Modificación diferida
- Modificación inmediata
- Modificaciones no comprometidas
- Paginación en la sombra
  - Tabla de páginas
  - Tabla de páginas actual
  - Tabla de páginas sombra
- Pérdida del almacenamiento no volátil
- Pestillos
- Puntos de revisión
- Puntos de revisión difusos
- Recogida de basura
- Recuperación basada en el registro histórico
- Recuperación con transacciones concurrentes
  - Retroceso de transacciones
  - Puntos de revisión difusos
  - Recuperación al reiniciar
- Registro actualizar del registro histórico
- Registro de escritura anticipada (REA)
- Registro histórico
- Registro histórico con memoria intermedia
- Registros del registro histórico
- Repetición de la historia
- Sistema operativo y gestión de la memoria intermedia
- Sistemas remotos de copia de seguridad
  - Sitio principal
  - Sitio remoto copia de seguridad
  - Sitio secundario
- Supuesto de fallo-parada
- Técnica de recuperación avanzada
  - Operación deshacer física
  - Operación deshacer lógica
  - Registro histórico físico
  - Registro histórico lógico
  - Operaciones lógicas
  - Retroceso de transacciones
  - Puntos de revisión
  - Recuperación al reiniciar
  - Fase rehacer
  - Fase deshacer

- Tiempo de compromiso
  - Uno seguro
  - Dos muy seguro
  - Dos seguro
- Tiempo de recuperación
- Tipos de almacenamiento
  - Almacenamiento volátil
  - Almacenamiento no volátil
  - Almacenamiento estable
- Transferencia del control
- Volcado de archivo
- Volcado difuso

EJERCICIOS

17.1. Explíquese la diferencia en cuanto al coste entre los tres tipos de almacenamiento: volátil, no volátil y estable.

17.2. No se puede implementar el almacenamiento estable.

a. Explíquese por qué no.

b. Explíquese cómo tratan este problema los sistemas de las bases de datos.

17.3. Compárense, en términos de facilidad de implementación y de sobrecarga, las versiones de modificación inmediata y modificación diferida de las técnicas de recuperación basadas en registro histórico.

17.4. Supóngase que un sistema utiliza modificación inmediata. Demuéstrese, con un ejemplo, cómo podría darse un estado inconsistente en la base de datos si no se escriben en almacenamiento estable los registros del registro histórico de una transacción antes de que el dato actualizado por la transacción se escriba a disco.

17.5. Explíquese el propósito del mecanismo de los puntos de revisión. ¿Con qué frecuencia deberían realizarse los puntos de revisión? Explíquese cómo afecta la frecuencia de los puntos de revisión:

- Al rendimiento del sistema cuando no ocurre ningún fallo
- Al tiempo que se tarda para recuperarse de una caída del sistema
- Al tiempo que se tarda para recuperarse de una caída del disco

17.6. Cuando el sistema se recupera después de una caída (véase el Apartado 17.6.4) construye una *lista-deshacer* y una *lista-rehacer*. Explíquese por qué deben procesarse en orden inverso los registros del registro histórico de las transacciones que se encuentran en la *lista-deshacer*, mientras que los registros del registro histórico correspondientes a las transacciones de la *lista-rehacer* se procesan hacia delante.

17.7. Compárense, en términos de facilidad de implementación y sobrecarga, el esquema de recuperación con paginación en la sombra con los esquemas de recuperación basados en registro histórico.

17.8. Considérese una base de datos compuesta por 10 bloques consecutivos en el disco (bloque 1, bloque 2, ..., bloque 10). Muéstrese el estado de la memoria intermedia y una posible ordenación física de los bloques después de las siguientes modificaciones, suponiendo que se utiliza paginación en la sombra, que la memoria intermedia en memoria principal sólo puede albergar tres bloques y que se usa la estrategia menos recientemente utilizada para gestionar la memoria intermedia.

```
leer bloque 3
leer bloque 7
leer bloque 5
leer bloque 3
leer bloque 1
modificar bloque 1
leer bloque 10
modificar bloque 5
```

17.9. Explíquese cómo el gestor de la memoria intermedia puede conducir a la base de datos a un estado inconsistente si algunos registros del registro histórico pertenecientes a un bloque no se escriben en almacenamiento estable antes de escribir en el disco el citado bloque.

17.10. Explíquense las ventajas del registro histórico lógico. Proporciónense ejemplos de una situación en la que sea preferible el registro histórico lógico frente al registro histórico físico y de una situación en la que sea preferible el registro histórico físico al registro histórico lógico.

17.11. Explíquense las razones por las que la recuperación en transacciones interactivas es más difícil de tratar que la recuperación en transacciones por lotes. ¿Existe una forma simple de tratar esta dificultad? (*Sugerencia*: considérese una transacción de un cajero automático por la que se retira dinero.)

17.12. A veces hay que deshacer una transacción después de que se haya comprometido porque se ejecutó erróneamente, debido por ejemplo a la introducción incorrecta de datos de un cajero.

a. Dese un ejemplo para demostrar que el uso de un mecanismo normal para deshacer esta transacción podría conducir a un estado inconsistente.

b. Una forma de manejar esta situación es llevar la base de datos a un estado anterior al compromiso de la transacción errónea (denominado recuperación a un instante). En este esquema se deshacen los efectos de las transacciones comprometidas después.

Sugiérase una modificación del mecanismo de recuperación avanzada para implementar la recuperación a un instante.

c. Las transacciones correctas se pueden volver a ejecutar lógicamente, pero no se pueden reejecutar usando sus registros del registro histórico. ¿Por qué?

17.13. En los lenguajes de programación persistentes no se realiza explícitamente el registro histórico de las modificaciones. Descríbase cómo pueden usarse las protecciones de acceso a las páginas que proporcionan los sistemas operativos modernos para crear imágenes anteriores y posteriores de las páginas que son modificadas. (*Sugerencia*: véase el Ejercicio 16.12.)

17.14. ARIES asume que hay espacio en cada página para un NSR. Al manejar objetos grandes que abarcan varias páginas, tales como archivos del sistema operativo, un objeto puede usar una página completa, sin dejar espacio para el NSR. Sugiérase una técnica para manejar esta situación; esta técnica debe soportar operaciones rehacer físicas pero no es necesario que soporte operaciones rehacer fisiológicas.

17.15. Explíquese la diferencia entre una caída del sistema y un «desastre».

17.16. Para cada uno de los siguientes requisitos identifíquese la mejor opción del grado de durabilidad en un sistema remoto de copia de seguridad.

a. Pérdida de datos que se debe evitar pero se puede tolerar alguna pérdida de disponibilidad.

b. El compromiso de transacciones se debe realizar rápidamente, incluso perdiendo algunas transacciones comprometidas en caso de desastre.

c. Se requiere un alto grado de disponibilidad y durabilidad, pero es aceptable un mayor tiempo de ejecución para el protocolo de compromiso de transacciones.

NOTAS BIBLIOGRÁFICAS

El libro de Gray y Reuter [1993] constituye una excelente fuente de información sobre la recuperación incluyendo interesantes implementaciones y detalles históricos. El libro de Bernstein et al. [1987] es una fuente de información sobre el control de concurrencia y recuperación.

Davies [1973] y Bjork [1973] son dos de los primeros documentos que presentan trabajos teóricos en el campo de la recuperación. Otro trabajo pionero en este campo es el de Chandy et al. [1975], el cual describe modelos analíticos para el retroceso y las estrategias de recuperación en los sistemas de bases de datos.

En Gray et al. [1981b] se presenta una visión de conjunto del esquema de recuperación de System R. El mecanismo de paginación en la sombra de System R se describe en Lorie [1977]. En Gray [1978], Lindsay et al. [1980] y Verhofstad [1978] pueden encontrarse guías de aprendizaje y visiones de conjunto sobre varias técnicas de recuperación para sistemas de bases de datos. Los conceptos de punto de revisión difuso y volcado difuso se describen en Lindsay et al. [1980]. Haerder y Reuter [1983] ofrece una amplia presentación de los principios de la recuperación.

El estado del arte de los métodos de recuperación se ilustra mejor con el método de recuperación ARIES, descrito en Mohan et al. [1992] y en Mohan [1990b]. ARIES y sus variantes se usan en varios productos de bases de datos, incluyendo DB2 de IBM y Microsoft SQL Server. La recuperación en Oracle se describe en Lahiri et al. [2001].

Mohan y Levine [1992] y Mohan [1993] proporcionan técnicas de recuperación especiales para estructuras con índices; Mohan y Narang [1994] describen técnicas de recuperación para arquitecturas cliente-servidor, mientras que Mohan y Narang [1991] y Mohan y Narang [1992] describen técnicas de recuperación para arquitecturas de bases de datos paralelas.

King et al. [1991] y Polyzois y García-Molina [1994] consideran las copias de seguridad remotas para recuperación de desastres (pérdida completa de un componente del sistema informático a causa de, por ejemplo, un incendio, una inundación o un terremoto).

En el Capítulo 24 se encuentran referencias sobre las transacciones de larga duración y los aspectos de recuperación relacionados.



ARQUITECTURA DE LOS SISTEMAS DE BASES DE DATOS

La arquitectura de los sistemas de bases de datos está enormemente influenciada por el sistema informático subyacente en el que se ejecuta el sistema de bases de datos. Los sistemas de bases de datos pueden ser centralizados, o cliente-servidor, donde una máquina que hace de servidor ejecuta trabajos de múltiples máquinas clientes. Los sistemas de bases de datos también pueden diseñarse para explotar las arquitecturas paralelas de computadoras. Las bases de datos distribuidas abarcan muchas máquinas separadas geográficamente.

El Capítulo 18 comienza tratando las arquitecturas de los sistemas de bases de datos que se ejecutan en sistemas servidores, los cuales se utilizan en arquitecturas centralizadas y cliente-servidor. En este capítulo se tratan los diferentes procesos que juntos implementan la funcionalidad de la base de datos. Después, se estudian las arquitecturas paralelas de computadoras y las arquitecturas paralelas de bases de datos diseñadas para diferentes tipos de computadoras paralelas. Finalmente, el capítulo trata asuntos arquitectónicos para la construcción de un sistema distribuido de bases de datos.

El Capítulo 19 presenta varias cuestiones que surgen en una base de datos distribuida, y describe cómo tratar cada cuestión. Estas cuestiones incluyen cómo almacenar datos, cómo asegurar la atomicidad de las transacciones que se ejecutan en varios emplazamientos, cómo realizar el control de la concurrencia y cómo proporcionar alta disponibilidad ante la presencia de fallos. En este capítulo también se estudian el procesamiento distribuido de consultas y los sistemas de directorio.

El Capítulo 20 describe cómo varias acciones de una base de datos, en particular el procesamiento de consultas, se pueden implementar para explotar el procesamiento paralelo.

CAPÍTULO

18 ARQUITECTURAS DE LOS SISTEMAS DE BASES DE DATOS

La arquitectura de un sistema de bases de datos está influenciada en gran medida por el sistema informático subyacente en el que se ejecuta, en particular por aspectos de la arquitectura de la computadora como la conexión en red, el paralelismo y la distribución:

- La conexión en red de varias computadoras permite que algunas tareas se ejecuten en un sistema servidor y que otras se ejecuten en los sistemas clientes. Esta división de trabajo ha conducido al desarrollo de *sistemas de bases de datos cliente-servidor*.
- El procesamiento paralelo dentro de una computadora permite acelerar las actividades del sistema de base de datos, proporcionando a las transacciones unas respuestas más rápidas así como la capacidad de ejecutar más transacciones por segundo. Las consultas pueden procesarse de manera que se explote el paralelismo ofrecido por el sistema informático subyacente. La necesidad del procesamiento paralelo de consultas ha conducido al desarrollo de los *sistemas de bases de datos paralelos*.
- La distribución de datos a través de las distintas sedes o departamentos de una organización permite que estos datos residan donde han sido generados o donde son más necesarios, pero continuar siendo accesibles desde otros lugares o departamentos diferentes. El hecho de guardar varias copias de la base de datos en diferentes sitios permite que puedan continuar las operaciones sobre la base de datos aunque algún sitio se vea afectado por algún desastre natural como una inundación, un incendio o un terremoto. Se han desarrollado los *sistemas distribuidos de bases de datos* para manejar datos distribuidos geográfica o administrativamente a lo largo de múltiples sistemas de bases de datos.

En este capítulo se estudia la arquitectura de los sistemas de bases de datos comenzando con los tradicionales sistemas centralizados y tratando, más adelante, los sistemas de bases de datos cliente-servidor, paralelos y distribuidos.

18.1. ARQUITECTURAS CENTRALIZADAS Y CLIENTE-SERVIDOR

Los sistemas de bases de datos centralizados son aquellos que se ejecutan en un único sistema informático sin interaccionar con ninguna otra computadora. Tales sistemas comprenden el rango desde los sistemas de bases de datos monousuario ejecutándose en computadoras personales hasta los sistemas de bases de datos de alto rendimiento ejecutándose en grandes sistemas. Por otro lado, los sistemas cliente-servidor tienen su funcionalidad dividida entre el sistema servidor y múltiples sistemas clientes.

18.1.1. Sistemas centralizados

Una computadora moderna de propósito general consiste en una o unas pocas unidades centrales de procesamiento y un número determinado de controladores para los dispositivos que se encuentran conectados a través de un bus común, el cual proporciona acceso a la memoria compartida (Figura 18.1). Las UCP (unidades centrales de procesamiento) poseen memorias caché locales donde se almacenan copias de ciertas partes de la memoria para acelerar el acceso a los datos. Cada controlador de dispositivo se encarga de un tipo específico de dispositivos (por ejemplo, una unidad de disco, una tarjeta de sonido o un monitor). Las UCP y los controladores de dispositivos pueden ejecutarse concurrentemente compitiendo así por el acceso a la memoria. La memoria caché reduce la disputa por el acceso a la memoria, ya que la UCP necesita acceder a la memoria compartida un número de veces menor.

Se distinguen dos formas de utilizar las computadoras: como sistemas monousuario o multiusuario. En la primera categoría están las computadoras personales y las estaciones de trabajo. Un **sistema monousuario** típico es una unidad de sobremesa utilizada por una única persona que dispone de una sola UCP, de uno o dos discos fijos y que trabaja con un sistema operativo que sólo permite un único usuario. Por el contrario, un **sistema**

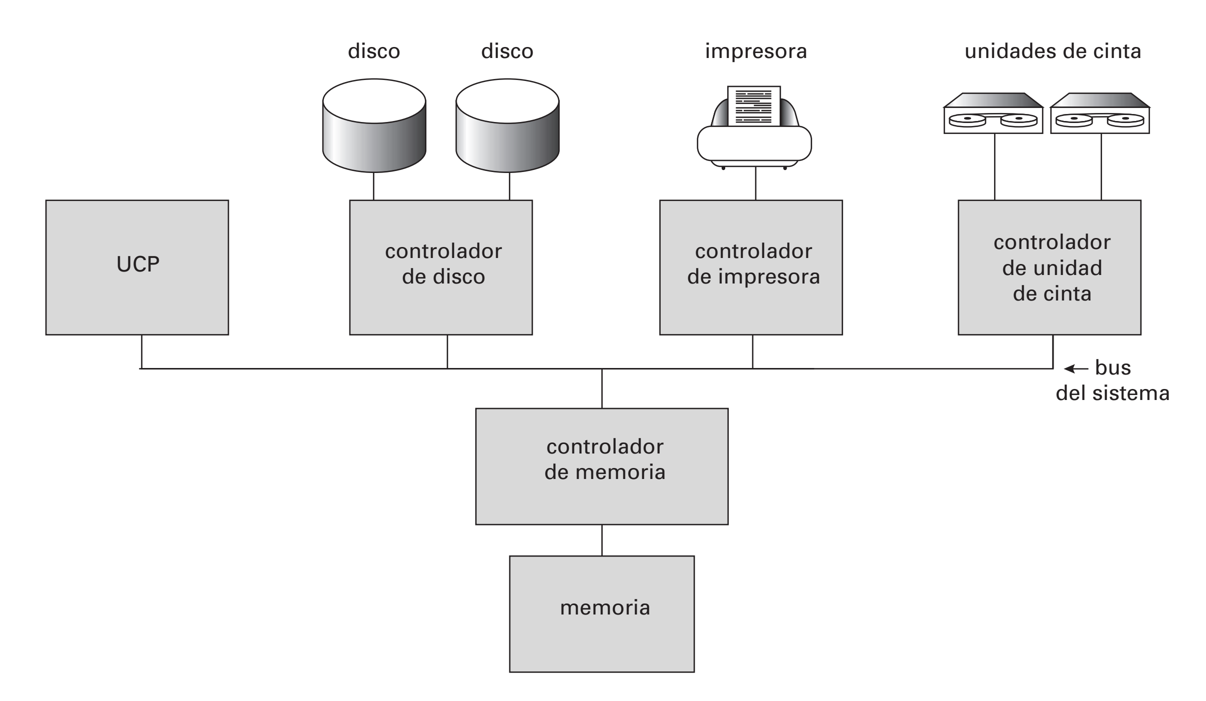


FIGURA 18.1. Un sistema informático centralizado.

multiusuario típico tiene más discos y más memoria, puede disponer de varias UCP y trabaja con un sistema operativo multiusuario. Se encarga de dar servicio a un gran número de usuarios que están conectados al sistema a través de terminales.

Normalmente, los sistemas de bases de datos diseñados para funcionar sobre sistemas monousuario no suelen proporcionar muchas de las facilidades que ofrecen los sistemas multiusuario. En particular no tienen control de concurrencia, que no es necesario cuando solamente un usuario puede generar modificaciones. Las facilidades de recuperación en estos sistemas o no existen o son primitivas; por ejemplo, realizar una copia de seguridad de la base de datos antes de cualquier modificación. La mayoría de estos sistemas no admiten SQL y proporcionan un lenguaje de consulta muy simple que, en algunos casos, es una variante de QBE. En cambio, los sistemas de bases de datos diseñados para sistemas multiusuario soportan todas las características de las transacciones que se han estudiado antes.

Aunque hoy en día las computadoras de propósito general tienen varios procesadores, utilizan **paralelismo de grano grueso**, disponiendo de unos pocos procesadores (normalmente dos o cuatro) que comparten la misma memoria principal. Las bases de datos que se ejecutan en tales máquinas habitualmente no intentan dividir una consulta simple entre los distintos procesadores, sino que ejecuta cada consulta en un único procesador posibilitando la concurrencia de varias consultas. Así, estos sistemas soportan una mayor productividad, es decir, permiten ejecutar un mayor número de transacciones por segundo, a pesar de que cada transacción individualmente no se ejecute más rápido.

Las bases de datos diseñadas para las máquinas monoprocesador ya disponen de multitarea permitiendo que varios procesos se ejecuten a la vez en el mismo procesador, usando tiempo compartido, mientras que de cara al usuario parece que los procesos se están ejecutando en paralelo. De esta manera, desde un punto de vista lógico, las máquinas paralelas de grano grueso parecen ser idénticas a las máquinas monoprocesador, y pueden adaptarse fácilmente los sistemas de bases de datos diseñados para máquinas de tiempo compartido para que puedan ejecutarse sobre máquinas paralelas de grano grueso.

Por el contrario, las máquinas **paralelas de grano fino** tienen un gran número de procesadores y los sistemas de bases de datos que se ejecutan sobre ellas intentan hacer paralelas las tareas simples (consultas, por ejemplo) que solicitan los usuarios. En el Apartado 18.3 se estudia la arquitectura de los sistemas de bases de datos paralelos.

18.1.2. Sistemas cliente-servidor

Como las computadoras personales son ahora más rápidas, más potentes y más baratas, los sistemas se han ido distanciando de la arquitectura centralizada. Los terminales conectados a un sistema central han sido suplantados por computadoras personales. De igual forma, la interfaz de usuario, que solía estar gestionada directamente por el sistema central, está pasando a ser gestionada, cada vez más, por las computadoras personales. Como consecuencia, los sistemas centralizados actúan hoy como **sistemas servidores** que satisfacen las peticiones generadas por los *sistemas clientes*. En la Figura 18.2 se representa la estructura general de un sistema cliente-servidor.

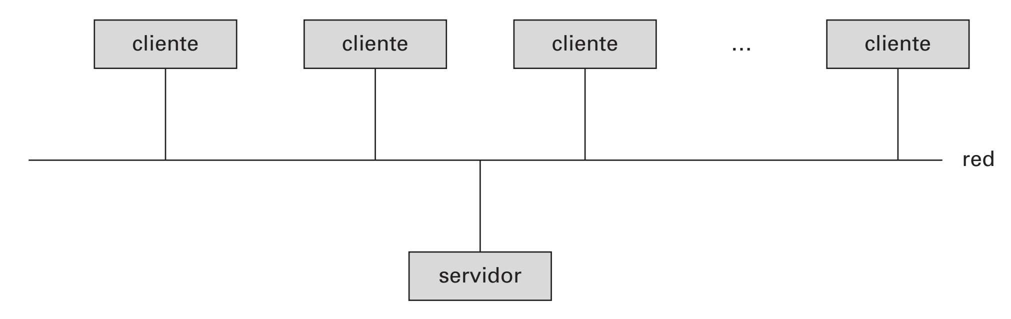


FIGURA 18.2. Estructura general de un sistema cliente-servidor.

Como se muestra en la Figura 18.3, la funcionalidad de una base de datos se puede dividir a grandes rasgos en dos partes: la parte visible al usuario y el sistema subyacente. El sistema subyacente gestiona el acceso a las estructuras, la evaluación y optimización de consultas, el control de concurrencia y la recuperación. La parte visible al usuario de un sistema de base de datos está formado por herramientas como formularios, diseñadores de informes y facilidades gráficas de interfaz de usuario. La interfaz entre la parte visible al usuario y el sistema subyacente puede ser SQL o una aplicación.

Las normas como *ODBC* y *JDBC*, que se vieron en el Capítulo 4, se desarrollaron para hacer de interfaz entre clientes y servidores. Cualquier cliente que utilice interfaces ODBC o JDBC puede conectarse a cualquier servidor que proporcione esta interfaz.

En las primeras generaciones de sistemas de bases de datos, la carencia de tales normas hacía que fuera necesario que la interfaz visible y el sistema subyacente fueran proporcionados por el mismo distribuidor de software. Con el aumento de las interfaces estándares, a menudo diferentes distribuidores proporcionan la interfaz visible al usuario y el servidor del sistema subyacente. Las *herramientas de desarrollo de aplicaciones* se utilizan para construir interfaces de usuario; proporcionan herramientas gráficas que se pueden utilizar para construir interfaces sin programar. Algunas de las herramientas de desarrollo de aplicaciones más famosas son PowerBuilder, Magic y Borland Delphi; Visual Basic también se utiliza bastante en el desarrollo de aplicaciones.

Además, ciertas aplicaciones como las hojas de cálculo y los paquetes de análisis estadístico utilizan la interfaz cliente-servidor directamente para acceder a los datos del servidor subyacente. De hecho, proporcionan interfaces visibles especiales para diferentes tareas.

Algunos sistemas de procesamiento de transacciones proporcionan una interfaz de **llamada a procedimientos remotos para transacciones** para conectar los clientes con el servidor. Estas llamadas aparecen para el programador como llamadas normales a procedimientos, pero todas las llamadas a procedimientos remotos hechas desde un cliente se engloban en una única transacción al servidor final. De este modo, si la transacción se cancela, el servidor puede deshacer los efectos de las llamadas a procedimientos remotos individuales.

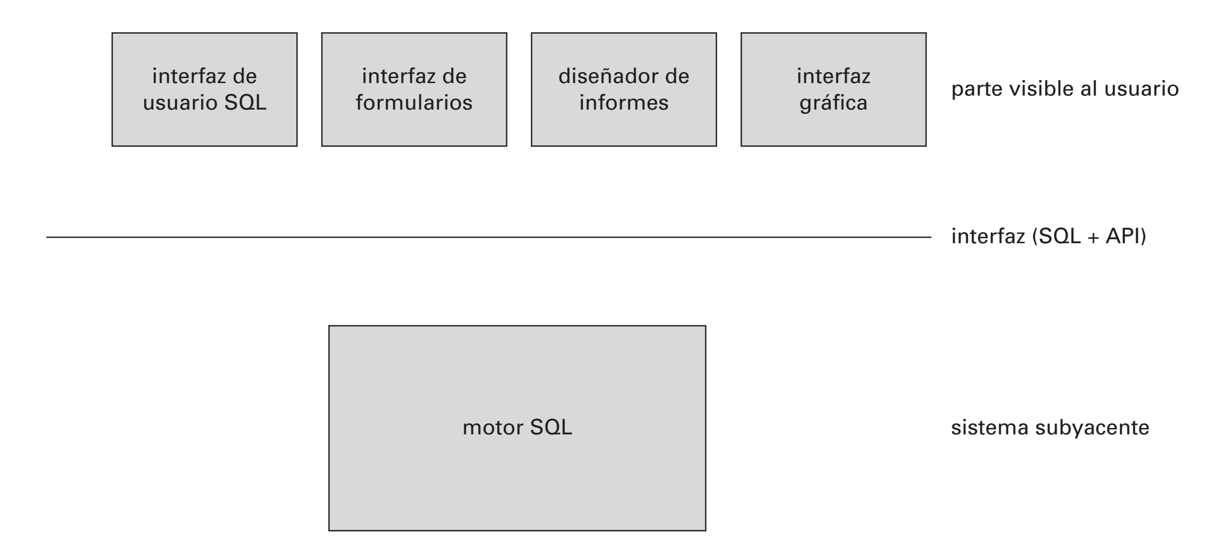


FIGURA 18.3. Funcionalidades de la parte visible al usuario y del sistema subyacente.

18.2. ARQUITECTURAS DE SISTEMAS SERVIDORES

Los sistemas servidores pueden dividirse en servidores de transacciones y servidores de datos.

- Los sistemas **servidores de transacciones**, también llamados sistemas **servidores de consultas**, proporcionan una interfaz a través de la cual los clientes pueden enviar peticiones para realizar una acción que el servidor ejecutará y cuyos resultados se devolverán al cliente. Normalmente, las máquinas cliente envían las transacciones a los sistemas servidores, lugar en el que estas transacciones se ejecutan, y los resultados se devuelven a los clientes que son los encargados de visualizar los datos. Las peticiones se pueden especificar utilizando SQL o mediante la interfaz de una aplicación especializada.
- Los **sistemas servidores de datos** permiten a los clientes interaccionar con los servidores realizando peticiones de lectura o modificación de datos en unidades tales como archivos o páginas. Por ejemplo, los servidores de archivos proporcionan una interfaz de sistema de archivos a través de la cual los clientes pueden crear, modificar, leer y borrar archivos. Los servidores de datos de los sistemas de bases de datos ofrecen muchas más funcionalidades; soportan unidades de datos de menor tamaño que los archivos, como páginas, tuplas u objetos. Proporcionan facilidades de indexación de los datos, así como facilidades de transacción de modo que los datos nunca se quedan en un estado inconsistente si falla una máquina cliente o un proceso.

De éstas, la arquitectura del servidor de transacciones es, con mucho, la arquitectura más ampliamente utilizada. En los Apartados 18.2.1 y 18.2.2 se desarrollarán las arquitecturas de los servidores de transacciones y de los servidores de datos.

18.2.1. Estructura de procesos del servidor de transacciones

Hoy en día, un sistema servidor de transacciones típico consiste en múltiples procesos accediendo a los datos en una memoria compartida, como en la Figura 18.4. Los procesos que forman parte del sistema de bases de datos incluyen:

- **Procesos servidor:** son procesos que reciben consultas del usuario (transacciones), las ejecutan, y devuelven los resultados. Las consultas deben enviarse a los procesos servidor desde la interfaz de usuario, o desde un proceso de usuario que ejecuta SQL incorporado, o a través de JDBC, ODBC o protocolos similares. Algunos sistemas de bases de datos utilizan un proceso distinto para cada sesión de usuario, y algunas utilizan un único proceso de la base de datos para todas las sesiones del usuario, pero con múltiples hebras de forma que se pueden ejecutar concurrentemente múltiples consultas. (Una **hebra** es como un proceso, pero varias hebras se ejecutan como parte del mismo proceso, y todas las hebras dentro de un proceso se ejecutan en el mismo espacio de memoria virtual. Dentro de un proceso se pueden ejecutar concurrentemente múltiples hebras.) Algunos sistemas de bases de datos utilizan una arquitectura híbrida, con procesos múltiples, cada uno de ellos con varias hebras.
- **Proceso gestor de bloqueos:** este proceso implementa una función de gestión de bloqueos que incluye concesión de bloqueos, liberación de bloqueos y detección de interbloqueos.
- **Proceso escritor de bases de datos:** hay uno o más procesos que vuelcan al disco los bloques de memoria intermedia modificados de forma continua.
- **Proceso escritor del registro:** este proceso genera entradas del registro en el almacenamiento estable a partir de la memoria intermedia del registro. Los procesos servidor simplifican la adición de entradas a la memoria intermedia del registro en memoria compartida y, si es necesario forzar la escritura del registro, le piden al proceso escritor del registro que vuelque las entradas del registro.
- **Proceso punto de revisión:** este proceso realiza periódicamente puntos de revisión.
- **Proceso monitor de proceso:** este proceso observa otros procesos y, si cualquiera de ellos falla, realiza acciones de recuperación para el proceso, tales como cancelar cualquier transacción que estuviera ejecutando el proceso fallido, y reinicia el proceso.

La memoria compartida contiene todos los datos compartidos, como:

- Grupo de memorias intermedias
- Tabla de bloqueos
- Memoria intermedia del registro, que contiene las entradas del registro que esperan a ser volcadas en el almacenamiento estable
- Planes de consulta en caché, que se pueden reutilizar si se envía de nuevo la misma consulta

Todos los procesos de la base de datos pueden acceder a los datos de la memoria compartida. Ya que múltiples procesos pueden leer o realizar actualizaciones en las estructuras de datos en memoria compartida, debe haber un mecanismo que asegure que sólo uno de ellos está modificando una estructura de datos en un momento dado, y que ningún proceso está leyendo una estructura de datos mientras otros la escriben. Tal

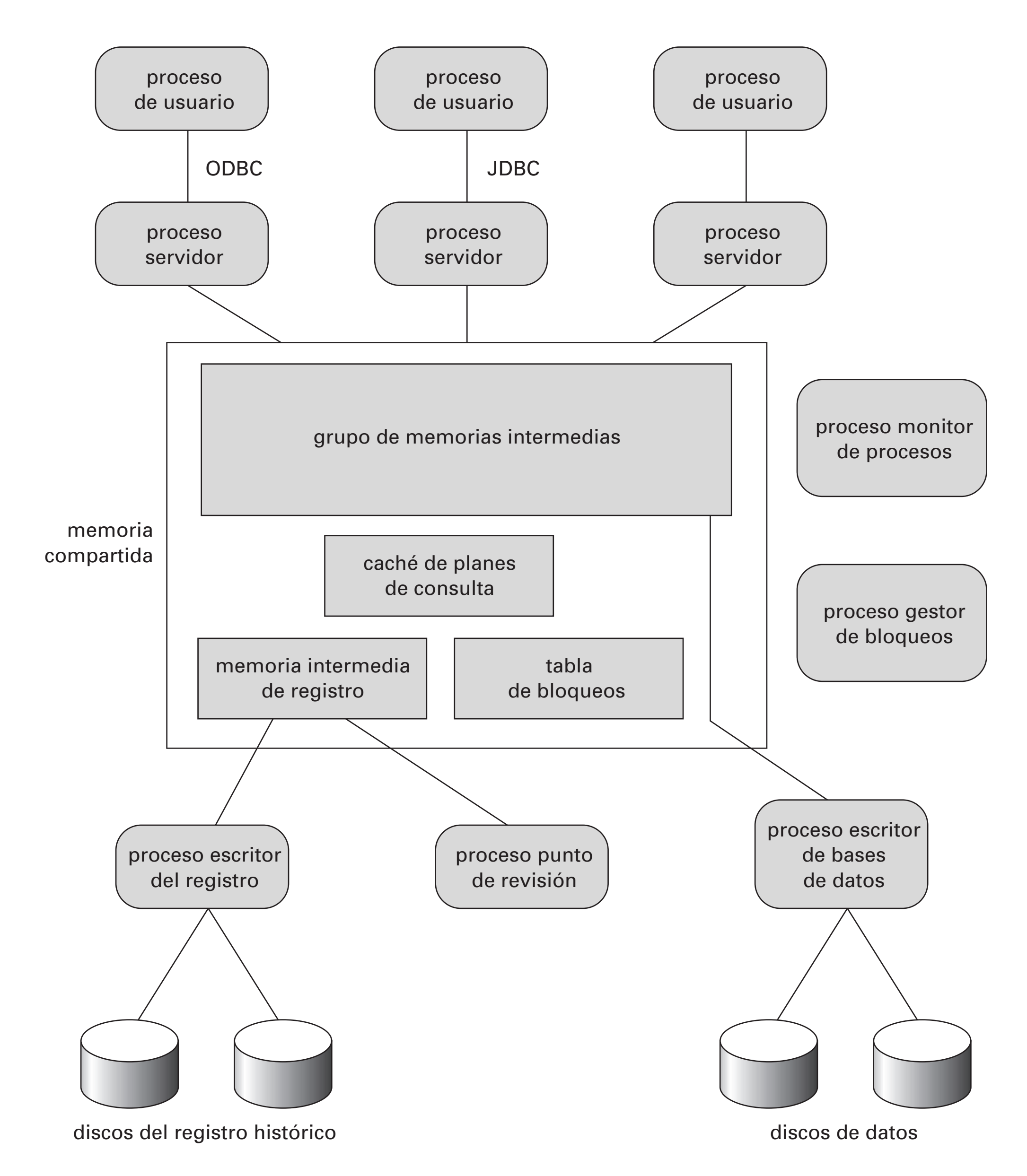


FIGURA 18.4. Estructura de la memoria compartida y de los procesos.

exclusión mutua se puede implementar por medio de funciones del sistema operativo llamadas semáforos. Implementaciones alternativas, con menos sobrecargas, utilizan **instrucciones atómicas** especiales soportadas por el hardware de la computadora; un tipo de instrucción atómica comprueba una posición de la memoria y la establece a uno automáticamente. Se pueden encontrar más detalles sobre la exclusión mutua en cualquier libro de texto de un sistema operativo estándar. Los mecanismos de exclusión mutua también se utilizan para implementar pestillos.

Para evitar la sobrecarga del paso de mensajes, en muchos sistemas de bases de datos los procesos servidor implementan el bloqueo actualizando directamente la tabla de bloqueos (que está en memoria compartida), en lugar de enviar mensajes de solicitud de bloqueo a un proceso administrador de bloqueos. El procedimiento de solicitud de bloqueos ejecuta las acciones que realizaría el proceso administrador de bloqueos para procesar una solicitud de bloqueo. Las acciones de la solicitud y la liberación de bloqueos son como las del Apartado 16.1.4, pero con dos diferencias significativas:

- Dado que varios procesos servidor pueden acceder a la memoria compartida, se asegurará la exclusión mutua en la tabla de bloqueos.
- Si no se puede obtener un bloqueo inmediatamente a causa de un conflicto de bloqueos, el código de la solicitud de bloqueo sigue observando la tabla de bloqueos hasta percatarse de que se ha concedido el bloqueo. El código de liberación de bloqueo actualiza la tabla de bloqueos para indicar a qué proceso se le ha concedido el bloqueo.

Para evitar repetidas comprobaciones de la tabla de bloqueos, el código de solicitud de bloqueo puede utilizar los semáforos del sistema operativo para esperar una notificación de una concesión de bloqueo. El código de liberación de bloqueo debe utilizar entonces el mecanismo de semáforos para notificar a las transacciones que están esperando que sus bloqueos hayan sido concedidos.

Incluso si el sistema gestiona las solicitudes de bloqueo por medio de memoria compartida, aún utiliza el proceso administrador de bloqueos para la detección de interbloqueos.

18.2.2. Servidores de datos

Los sistemas servidores de datos se utilizan en redes de área local en las que se alcanza una alta velocidad de conexión entre los clientes y el servidor, las máquinas clientes son comparables al servidor en cuanto a poder de procesamiento y se ejecutan tareas de cómputo intensivo. En este entorno tiene sentido enviar los datos a las máquinas clientes, realizar allí todo el procesamiento (que puede durar un tiempo) y después enviar los datos de vuelta al servidor. Nótese que esta arquitectura necesita que los clientes posean todas las funcionalidades del sistema subyacente. Las arquitecturas de los servidores de datos se han hecho particularmente populares en los sistemas de bases de datos orientadas a objetos.

En esta arquitectura surgen algunos aspectos interesantes, ya que el coste en tiempo de comunicación entre el cliente y el servidor es alto comparado al de acceso a una memoria local (milisegundos frente a menos de 100 nanosegundos).

- **Envío de páginas o envío de elementos.** La unidad de comunicación de datos puede ser de grano grueso, como una página, o de grano fino, como una tupla (o, en el contexto de los sistemas de bases de datos orientados a objetos, un objeto). Se usará el término **elemento** para referirse tanto a tuplas como a objetos.

  Si la unidad de comunicación de datos es un único elemento, la sobrecarga por la transferencia de mensajes es alta comparada con el número de datos transmitidos. En vez de hacer esto, cuando se necesita un elemento, cobra sentido la idea de enviar junto a aquél otros elementos que probablemente vayan a ser utilizados en un futuro próximo. Se denomina **preextracción** a la acción de buscar y enviar elementos antes de que sea estrictamente necesario. Si varios elementos residen en un página, el envío de páginas puede considerarse como una forma de preextracción, ya que, cuando un proceso desee acceder a un único elemento de la página, se enviarán todos los elementos de esa página.
- **Bloqueo.** La concesión del bloqueo de los elementos de datos que el servidor envía a los clientes la realiza habitualmente el propio servidor. Un inconveniente del envío de páginas es que los clientes pueden recibir bloqueos de grano grueso: el bloqueo de una página bloquea implícitamente a todos los elementos que residan en ella. El cliente adquiere implícitamente bloqueos sobre todos los elementos preextraídos incluso aunque no esté accediendo a algunos de ellos. De esta forma, puede detenerse innecesariamente el procesamiento de otros clientes que necesiten bloquear esos elementos. Se han propuesto algunas técnicas para la **liberación de bloqueos** en las que el servidor puede pedir a los clientes que le devuelvan el control sobre los bloqueos de los elementos preextraídos. Si el cliente no necesita el elemento preextraído puede devolver los bloqueos sobre ese elemento al servidor para que éstos puedan ser asignados a otros clientes.
- **Caché de datos.** Los datos que se envían al cliente en favor de una transacción se pueden **alojar en una caché** del cliente incluso una vez completada la transacción, si dispone de suficiente espacio de almacenamiento libre. Las transacciones sucesivas en el mismo cliente pueden hacer uso de los datos en caché. Sin embargo, se presenta el problema de la **coherencia de caché**: si una transacción encuentra los datos en la caché, debe asegurarse de que esos datos están al día, ya que, después de haber sido almacenados en la caché, pueden haber sido modificados por otro cliente. Así, debe establecerse una comunicación con el servidor para comprobar la validez de los datos y poder adquirir un bloqueo sobre ellos.
- **Caché de bloqueos.** Los bloqueos también pueden ser almacenados en la memoria caché del cliente si la utilización de los datos está prácticamente dividida entre los clientes, de manera que un cliente rara vez necesita datos que están siendo utilizados por otros clientes. Supóngase que se encuentran en la memoria caché tanto el elemento de datos que se busca como el bloqueo requerido para acceder al mismo. Entonces, el cliente puede acceder al elemento de datos sin necesidad de comunicar nada al servidor. No obstante, el servidor debe seguir el rastro de los bloqueos en caché; si un cliente solicita un bloqueo al servidor, éste debe **comunicar** a todos los bloqueos sobre el elemento de datos que se encuentren en las memorias caché de otros clientes. La tarea se vuelve más complicada cuando se tienen en cuenta los posibles fallos de la máquina. Esta técnica se diferencia de la liberación de bloqueos en que la caché de bloqueo se realiza a través de transacciones; de otra forma, las dos técnicas serían similares.

Las referencias bibliográficas proporcionan más información sobre los sistemas cliente-servidor de bases de datos.

18.3. SISTEMAS PARALELOS

Los sistemas paralelos mejoran la velocidad de procesamiento y de E/S mediante la utilización de UCP y discos en paralelo. Cada vez son más comunes las máquinas paralelas, lo que hace que cada vez sea más importante el estudio de los sistemas paralelos de bases de datos. La fuerza que ha impulsado a los sistemas paralelos de bases de datos ha sido la demanda de aplicaciones que han de manejar bases de datos extremadamente grandes (del orden de terabytes, esto es, 10^{12} bytes) o que tienen que procesar un número enorme de transacciones por segundo (del orden de miles de transacciones por segundo). Los sistemas de bases de datos centralizados o cliente-servidor no son suficientemente potentes para soportar tales aplicaciones.

En el procesamiento paralelo se realizan muchas operaciones simultáneamente mientras que en el procesamiento secuencial, los distintos pasos computacionales han de ejecutarse en serie. Una máquina paralela de **grano grueso** consiste en un pequeño número de potentes procesadores; una máquina **masivamente paralela** o de **grano fino** utiliza miles de procesadores más pequeños. Hoy en día, la mayoría de las máquinas de gama alta ofrecen un cierto grado de paralelismo de grano grueso: son comunes las máquinas con dos o cuatro procesadores. Las computadoras masivamente paralelas se distinguen de las máquinas paralelas de grano grueso porque son capaces de soportar un grado de paralelismo mucho mayor. Ya se encuentran en el mercado computadoras paralelas con cientos de UCP y discos.

Para medir el rendimiento de los sistemas de bases de datos existen dos medidas principales: (1) la **productividad**, número de tareas que pueden completarse en un intervalo de tiempo determinado, y (2) el **tiempo de respuesta**, cantidad de tiempo que necesita para completar una única tarea a partir del momento en que se envíe. Un sistema que procese un gran número de pequeñas transacciones puede mejorar la productividad realizando muchas transacciones en paralelo. Un sistema que procese transacciones largas puede mejorar el tiempo de respuesta así como la productividad realizando en paralelo las distintas subtareas de cada transacción.

18.3.1. Ganancia de velocidad y ampliabilidad

La ganancia de velocidad y la ampliabilidad son dos aspectos importantes en el estudio del paralelismo. La **ganancia de velocidad** se refiere a la ejecución en menos tiempo de una tarea dada mediante el incremento del grado de paralelismo. La **ampliabilidad** se refiere al manejo de transacciones más largas mediante el incremento del grado de paralelismo.

Considérese un sistema paralelo con un cierto número de procesadores y discos que está ejecutando una aplicación de base de datos. Supóngase ahora que se incrementa el tamaño del sistema añadiéndole más procesadores, discos y otros componentes. El objetivo es realizar el procesamiento de la tarea en un tiempo inversamente proporcional al número de procesadores y discos del sistema. Supóngase que el tiempo de ejecución de una tarea en la máquina más grande es T_G y que el tiempo de ejecución de la misma tarea en la máquina más pequeña es T_P. La ganancia de velocidad debida al paralelismo se define como T_P / T_G. Se dice que un sistema paralelo tiene una **ganancia de velocidad lineal** si la ganancia de velocidad es N cuando el sistema más grande tiene N veces más recursos (UCP, discos, etc.) que el sistema más pequeño. Si la ganancia de velocidad es menor que N se dice que el sistema tiene una **ganancia de velocidad sublineal**. En la Figura 18.5 se muestra la ganancia de velocidad lineal y sublineal.

La ampliabilidad está relacionada con la capacidad para procesar tareas más largas en el mismo tiempo mediante el incremento de los recursos del sistema. Sea Q una tarea y sea Q_N una tarea N veces más grande que Q. Supóngase que T_P es el tiempo de ejecución de la tarea Q en una máquina dada M_P y que T_G es el tiempo de ejecución de la tarea Q_N en una máquina paralela M_G, la cual es N veces más grande que M_P. La ampliabilidad se define como T_P / T_G. Se dice que el sistema paralelo M_G tiene una **ampliabilidad lineal** sobre la tarea Q si $T_P = T_G$. Si $T_G > T_P$ se dice que el sistema tiene una **ampliabilidad sublineal**. En la Figura 18.6 se muestra la ampliabilidad lineal y sublineal (donde los recursos aumentan proporcionalmente al tamaño del problema). La manera de medir el tamaño de las tareas da lugar a dos tipos de ampliabilidad relevantes en los sistemas paralelos de bases de datos:

- En la **ampliabilidad por lotes** aumenta el tamaño de la base de datos, y las tareas son trabajos más largos cuyos tiempos de ejecución dependen del tamaño de la base de datos. Recorrer una relación cuyo tamaño es proporcional al tamaño de la base de datos sería un ejemplo de tales tareas. Así, la

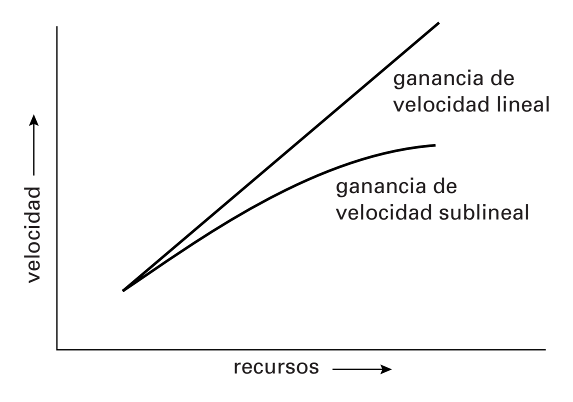


FIGURA 18.5. Ganancia de velocidad respecto al incremento de los recursos.

medida del tamaño del problema es el tamaño de la base de datos. La ampliabilidad por lotes también se utiliza en aplicaciones científicas tales como la ejecución de una consulta con una resolución *N* veces mayor o la realización de una simulación *N* veces más larga.

- En la **ampliabilidad de transacciones** aumenta la velocidad con la que se envían las transacciones a la base de datos y el tamaño de la base de datos crece proporcionalmente a la tasa de transacciones. Este tipo de ampliabilidad es el relevante en los sistemas de procesamiento de transacciones en los que las transacciones son modificaciones pequeñas –por ejemplo, un abono o retirada de fondos de una cuenta –y cuantas más cuentas se creen, más crece la tasa de transacciones. Este procesamiento de transacciones se adapta especialmente bien a la ejecución en paralelo, ya que las transacciones pueden ejecutarse concurrente e independientemente en procesadores distintos y cada transacción dura más o menos el mismo tiempo, aunque crezca la base de datos.

La ampliabilidad es normalmente el factor más importante para medir la eficiencia de un sistema paralelo de bases de datos. El objetivo del paralelismo en los sistemas de bases de datos suele ser asegurar que la ejecución del sistema continuará realizándose a una velocidad aceptable incluso en el caso de que aumente el tamaño de la base de datos o el número de transacciones. El incremento de la capacidad del sistema mediante el incremento del paralelismo proporciona a una empresa un modo de crecimiento más suave que el de reemplazar un sistema centralizado por una máquina más rápida (suponiendo incluso que esta máquina existiera). Sin embargo, cuando se utiliza la ampliabilidad debe atenderse también a los valores del rendimiento absoluto; una máquina con un ampliabilidad lineal puede tener un rendimiento más bajo que otra con ampliabilidad sublineal simplemente porque la última sea mucho más rápida que la primera.

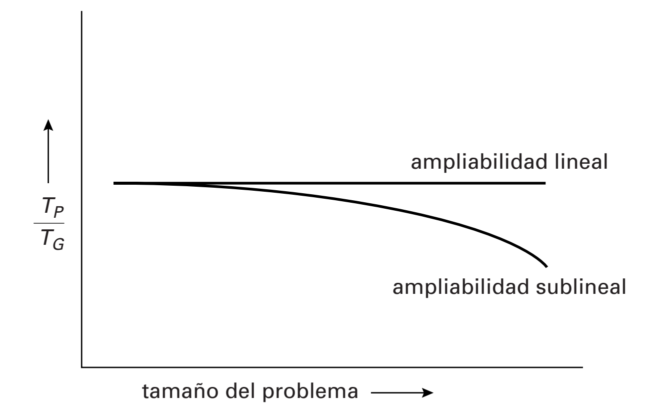


FIGURA 18.6. Ampliabilidad respecto al crecimiento del tamaño del problema.

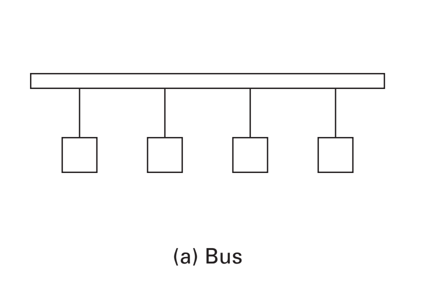
Existen algunos factores que trabajan en contra de la eficiencia del paralelismo y pueden atenuar tanto la ganancia de velocidad como la ampliabilidad.

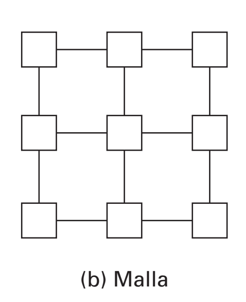
- **Costes de inicio.** El inicio de un único proceso lleva asociado un coste. En una operación paralela compuesta por miles de procesos el *tiempo de inicio* puede llegar a ser mucho mayor que el tiempo real de procesamiento, lo que influye negativamente en la ganancia de velocidad.
- **Interferencia.** Como los procesos que se ejecutan en un sistema paralelo acceden con frecuencia a recursos compartidos, pueden sufrir un cierto retardo como consecuencia de la *interferencia* de cada nuevo proceso en la competencia con los procesos existentes por el acceso a los recursos más comunes, como el bus del sistema, los discos compartidos o incluso los bloqueos. Este fenómeno afecta tanto a la ganancia de velocidad como a la ampliabilidad.
- **Sesgo.** Al dividir cada tarea en un cierto número de pasos paralelos se reduce el tamaño del paso medio. Es más, el tiempo de servicio de la tarea completa vendrá determinado por el tiempo de servicio del paso más lento. Normalmente es difícil dividir una tarea en partes exactamente iguales, entonces se dice que la forma de distribución de los tamaños es *sesgada*. Por ejemplo, si se divide una tarea de tamaño 100 en 10 partes y la división está sesgada, puede haber algunas tareas de tamaño menor que 10 y otras de tamaño superior a 10; si el tamaño de una tarea fuera 20, la ganancia de velocidad que se obtendría al ejecutar las tareas en paralelo sólo valdría 5 en vez de lo que cabría esperarse, 10.

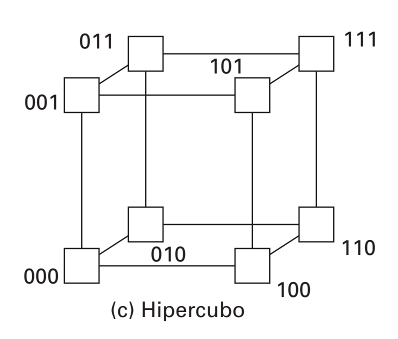
18.3.2. Redes de interconexión

Los sistemas paralelos están constituidos por un conjunto de componentes (procesadores, memoria y discos) que pueden comunicarse entre sí a través de una **red de interconexión**. La Figura 18.7 muestra tres tipos de redes de interconexión utilizados frecuentemente:

- **Bus.** Todos los componentes del sistema pueden enviar o recibir datos de un único bus de comunicaciones. Este tipo de interconexión se muestra en la Figura 18.7a. El bus puede ser una red Ethernet o una interconexión paralela. Las arquitecturas de bus trabajan bien para un pequeño número de procesadores. Sin embargo, como el bus sólo puede gestionar la comunicación de un único componente en cada momento, las arquitecturas de bus son menos apropiadas según aumenta el paralelismo.
- **Malla.** Los componentes se organizan como los nodos de una retícula de modo que cada componente está conectado con todos los nodos adyacentes. En una malla bidimensional cada nodo está conectado con cuatro nodos adyacentes, mientras


(a) Bus


(b) Malla



(c) Hipercubo

FIGURA 18.7. Redes de interconexión.

que en una malla tridimensional cada nodo está conectado con seis nodos adyacentes. La Figura 18.7b muestra una malla bidimensional. Los nodos entre los que no existe una conexión directa pueden comunicarse mediante el envío de mensajes a través de una secuencia de nodos intermedios que sí dispongan de conexión directa. A medida que aumenta el número de componentes también aumenta el número de enlaces de comunicación, por lo que la capacidad de comunicación, de una malla es mejor cuanto mayor es el paralelismo.

- **Hipercubo**. Se asigna a cada componente un número binario de modo que dos componentes tienen una conexión directa si sus correspondientes representaciones binarias difieren en un solo bit. Así, cada uno de los n componentes está conectado con otros $log(n)$ componentes. La Figura 18.7c muestra un hipercubo con 8 vértices. Puede demostrarse que, en un hipercubo, un mensaje de un componente puede llegar a cualquier otro componente de la red de interconexión atravesando a lo sumo $log(n)$ enlaces. Por el contrario, en una malla un componente puede estar a $2(\sqrt{n} - 1)$ enlaces de otros componentes (o a $\sqrt{n}$ enlaces de distancia si la malla de interconexión conecta entre sí los bordes opuestos). De esta manera, el retardo de la comunicación en un hipercubo es significativamente menor que en una malla.

18.3.3. Arquitecturas paralelas de bases de datos

Existen varios modelos de arquitecturas para las máquinas paralelas. En la Figura 18.8 se muestran algunos de los más importantes (en la figura, M quiere decir memoria, P procesador y los discos se dibujan como cilindros):

- **Memoria compartida.** Todos los procesadores comparten una memoria común (Figura 18.8a).
- **Disco compartido.** Todos los procesadores comparten un conjunto de discos común (Figura 18.8b). Algunas veces los sistemas de disco compartido se denominan **agrupaciones**.

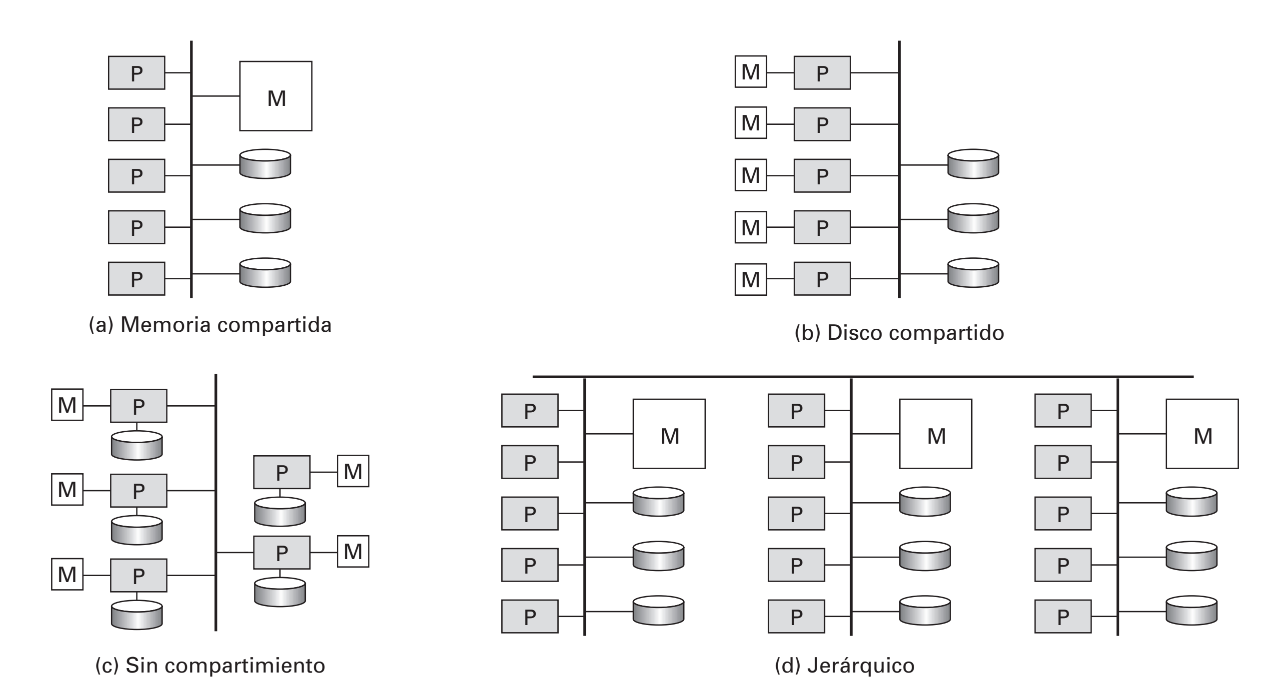


FIGURA 18.8. Arquitecturas paralelas de bases de datos.

- **Sin compartimiento**. Los procesadores no comparten ni memoria ni disco (Figura 18.8c).
- **Jerárquico**. Este modelo es un híbrido de las arquitecturas anteriores (Figura 18.8d).

En los Apartados 18.3.3.1 hasta el 18.3.3.4 se abordará cada uno de estos modelos.

Las técnicas utilizadas para acelerar el procesamiento de transacciones en sistemas servidores de datos, como la caché de datos y bloqueos y la liberación de bloqueos, tratadas en el Apartado 18.2.2, también se pueden utilizar en bases de datos paralelas de discos compartidos además de en bases de datos paralelas sin compartimiento. De hecho, son muy importantes para el procesamiento eficiente de transacciones en tales sistemas.

18.3.3.1. Memoria compartida

En una arquitectura de **memoria compartida** los procesadores y los discos tienen acceso a una memoria común, normalmente a través de un bus o de una red de interconexión. El beneficio de la memoria compartida es la extremada eficiencia en cuanto a la comunicación entre procesadores; cualquier procesador puede acceder a los datos de la memoria compartida sin necesidad de la intervención del software. Un procesador puede enviar mensajes a otros procesadores utilizando escrituras en la memoria de modo que la velocidad de envío es mucho mayor (normalmente es inferior a un microsegundo) que la que se alcanza con un mecanismo de comunicación. El inconveniente de las máquinas con memoria compartida es que la arquitectura no puede ir más allá de 32 o 64 procesadores porque el bus o la red de interconexión se convertirían en un cuello de botella (ya que está compartido por todos los procesadores). Llega un momento en el que no sirve de nada añadir más procesadores, ya que éstos emplean la mayoría de su tiempo esperando su turno para utilizar el bus y así poder acceder a la memoria.

Las arquitecturas de memoria compartida suelen dotar a cada procesador de una memoria caché muy grande para evitar las referencias a la memoria compartida siempre que sea posible. No obstante, en la caché no podrán estar todos los datos y no podrá evitarse el acceso a la memoria compartida. Además, las cachés necesitan mantener la coherencia; esto es, si un procesador realiza una escritura en una posición de memoria, los datos de dicha posición de memoria se deberían actualizar en o eliminar de cualquier procesador donde estuvieran los datos en caché. El mantenimiento de la coherencia de la caché aumenta la sobrecarga cuando aumenta el número de procesadores. Por estas razones las máquinas con memoria compartida no pueden extenderse llegado un punto; las máquinas actuales con memoria compartida no pueden soportar más de 64 procesadores.

18.3.3.2. Disco compartido

En el modelo de **disco compartido** todos los procesadores pueden acceder directamente a todos los discos a través de una red de interconexión, pero los procesadores tienen memorias privadas. Las arquitecturas de disco compartido ofrecen dos ventajas respecto de las de memoria compartida. Primero, el bus de la memoria deja de ser un cuello de botella, ya que cada procesador dispone de memoria propia. Segundo, esta arquitectura ofrece una forma barata para proporcionar una cierta **tolerancia ante fallos**: si falla un procesador (o su memoria) los demás procesadores pueden hacerse cargo de sus tareas, ya que la base de datos reside en los discos, a los cuales tienen acceso todos los procesadores. Como se describía en el Capítulo 11, utilizando una arquitectura RAID también puede conseguirse que el subsistema de discos sea tolerante ante fallos por sí mismo. La arquitectura de disco compartido tiene aceptación en bastantes aplicaciones.

El problema principal de los sistemas de discos compartidos es, de nuevo, la ampliabilidad. Aunque el bus de la memoria no es cuello de botella muy grande, la interconexión con el subsistema de discos es ahora el nuevo cuello de botella; esto es especialmente grave en situaciones en las que la base de datos realiza un gran número de accesos a los discos. Los sistemas de discos compartidos pueden soportar un mayor número de procesadores en comparación con los sistemas de memoria compartida, pero la comunicación entre los procesadores es más lenta (hasta unos pocos milisegundos si se carece de un hardware de propósito especial para comunicaciones), ya que se realiza a través de una red de interconexión.

Las agrupaciones DEC con Rdb constituyen uno de los primeros usuarios de la arquitectura de bases de datos de disco compartido (Rdb ahora es propiedad de Oracle y se denomina Oracle Rdb. Digital Equipment Corporation (DEC) es ahora propiedad de Compaq).

18.3.3.3. Sin compartimiento

En un sistema **sin compartimiento** cada nodo de la máquina consta de un procesador, memoria y uno o más discos. Los procesadores de un nodo pueden comunicarse con un procesador de otro nodo utilizando una red de interconexión de alta velocidad. Un nodo funciona como el servidor de los datos almacenados en los discos que posee. El modelo sin compartimiento salva el inconveniente de requerir que todas las operaciones de E/S vayan a través de una única red de interconexión, ya que las referencias a los discos locales son servidas por los discos locales de cada procesador; solamente van por la red las peticiones, los accesos a discos remotos y las relaciones de resultados. Es más, habitualmente las redes de interconexión para los sistemas sin compartimiento se diseñan para ser ampliables por lo que su capacidad de transmisión crece a medida que se añaden nuevos nodos. Como consecuencia, las arquitecturas sin compartimiento son más ampliables y pueden soportar con facilidad un gran número de procesadores. El principal inconveniente de los sistemas sin compartimiento es el coste de comunicación y de acceso a discos remotos, coste que es mayor que el que se produce

en las arquitecturas de memoria o disco compartido, ya que el envío de datos provoca la intervención del software en ambos extremos.

La máquina de base de datos Teradata fue uno de los primeros sistemas comerciales que utilizaron la arquitectura sin compartimiento de bases de datos. También se construyeron sobre arquitecturas sin compartimiento los prototipos de investigación Grace y Gamma.

18.3.3.4. Jerárquica

La **arquitectura jerárquica** combina las características de las arquitecturas de memoria compartida, de disco compartido y sin compartimiento. A alto nivel el sistema está formado por nodos que están conectados mediante una red de interconexión y que no comparten ni memoria ni discos. Así, el nivel más alto es una arquitectura sin compartimiento. Cada nodo del sistema podría ser en realidad un sistema de memoria compartida con algunos procesadores. Alternativamente, cada nodo podría ser un sistema de disco compartido y cada uno de estos sistemas de disco compartido podría ser a su vez un sistema de memoria compartida. De esta manera, un sistema podría construirse como una jerarquía con una arquitectura de memoria compartida con pocos procesadores en la base, en lo más alto una arquitectura sin compartimiento y quizá una arquitectura de disco compartido en el medio. En la Figura 18.8d se muestra una arquitectura jerárquica con nodos de memoria compartida conectados entre sí con una arquitectura sin compartimiento. Hoy en día los sistemas paralelos comerciales de bases de datos pueden ejecutarse sobre varias de estas arquitecturas.

Las arquitecturas de **memoria virtual distribuida**, en las que hay una única memoria compartida desde el punto de vista lógico pero hay varios sistemas de memoria disjuntos desde el punto de vista físico, han surgido tras varios intentos por reducir la complejidad de programación de los sistemas jerárquicos; se obtiene una única vista del área de memoria virtual de estas memorias disjuntas mediante un hardware de asignación de memoria virtual en conjunción con un software extra. Dado que las velocidades de acceso son diferentes, dependiendo de si la página está disponible localmente o no, esta arquitectura también se denomina **arquitectura de memoria no uniforme (NUMA, Nonuniform Memory Architecture)**.

18.4. SISTEMAS DISTRIBUIDOS

En un **sistema distribuido de bases** de datos se almacena la base de datos en varias computadoras. Varios medios de comunicación, como las redes de alta velocidad o las líneas telefónicas, son los que pueden poner en contacto las distintas computadoras de un sistema distribuido. No comparten ni memoria ni discos. Las computadoras de un sistema distribuido pueden variar en tamaño y función pudiendo abarcar desde las estaciones de trabajo a los grandes sistemas.

Dependiendo del contexto en el que se mencionen existen diferentes nombres para referirse a las computadoras que forman parte de un sistema distribuido, tales como **sitios** o **nodos**. Para enfatizar la distribución física de estos sistemas se usa principalmente el término **sitio**. En la Figura 18.9 se muestra la estructura general de un sistema distribuido.

Las principales diferencias entre las bases de datos paralelas sin compartimientos y las bases de datos distribuidas son que las bases de datos distribuidas normalmente se encuentran en varios lugares geográficos distintos, se administran de forma separada y poseen una interconexión más lenta. Otra gran diferencia es que en un sistema distribuido se dan dos tipos de transacciones, las locales y las globales. Una **transacción local** es aquella que accede a los datos del único sitio en el cual se inició la transacción. Por otra parte, una **transacción global** es aquella que, o bien accede a los datos situados en un sitio diferente de aquel en el que se inició la transacción, o bien accede a datos de varios sitios distintos.

Hay varias razones para construir sistemas distribuidos de bases de datos, incluyendo el compartimiento de los datos, la autonomía y la disponibilidad.

- **Datos compartidos.** La principal ventaja de construir un sistema distribuido de bases de datos es poder disponer de un entorno donde los usuarios puedan acceder desde una única ubicación a los datos que residen en otras ubicaciones. Por ejemplo, en un sistema de banca distribuida, donde cada sucursal almacena datos relacionados con dicha sucursal, es posible que un usuario de una de las sucursales acceda a los datos de otra sucursal. Sin esta capacidad, un usuario que quisiera transferir fondos de una sucursal a otra tendría que recurrir a algún mecanismo externo que pudiera enlazar los sistemas existentes.
- **Autonomía.** La principal ventaja de compartir datos por medio de distribución de datos es que cada ubicación es capaz de mantener un grado de control sobre los datos que se almacenan localmente. En un sistema centralizado, el administrador de bases de datos de la ubicación central controla la base de datos. En un sistema distribuido, existe un administrador de bases de datos global responsable de todo el sistema. Una parte de estas responsabilidades se delegan al administrador de bases de datos local de cada sitio. Dependiendo del diseño del sistema distribuido de bases de datos,

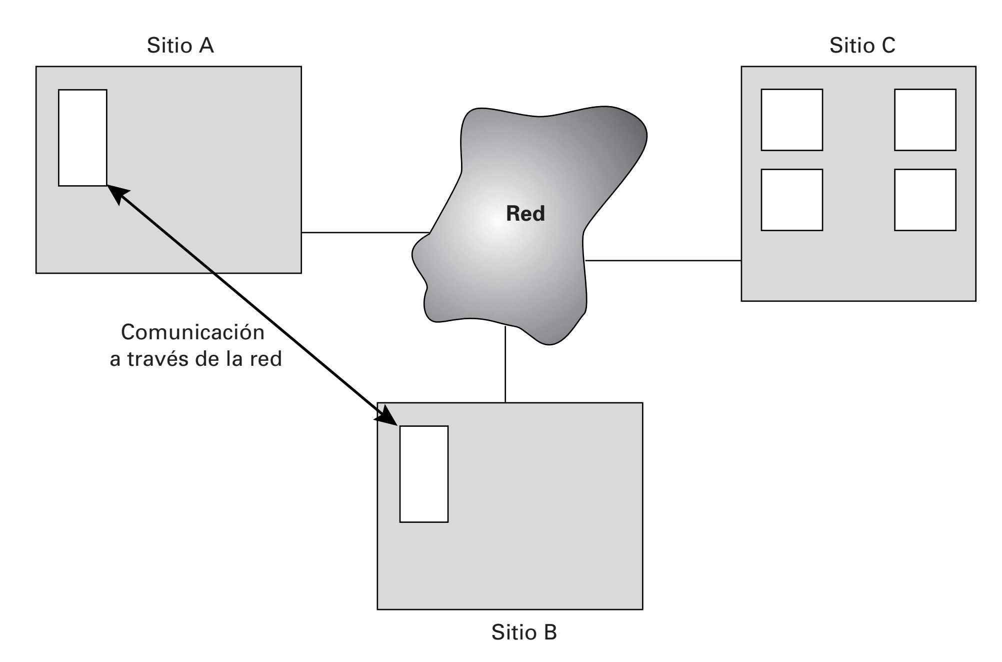


FIGURA 18.9. Un sistema distribuido.

cada administrador puede tener un grado diferente de **autonomía local**. La posibilidad de autonomía local es a menudo una de las grandes ventajas de las bases de datos distribuidas.

- **Disponibilidad.** Si un sitio de un sistema distribuido falla, los sitios restantes pueden seguir trabajando. En particular, si los elementos de datos están **replicados** en varios sitios, una transacción que necesite un elemento de datos en particular puede encontrarlo en varios sitios. De este modo, el fallo de un sitio no implica necesariamente la caída del sistema.

  El sistema puede detectar el fallo de un sitio, y pueden ser necesarias acciones apropiadas para recuperarse del fallo. El sistema no debe seguir utilizando los servicios del sitio que falló. Finalmente, cuando el sitio que falló se recupera o se repara, debe haber mecanismos disponibles para integrarlo sin problemas de nuevo en el sistema.

  Aunque la recuperación ante un fallo es más compleja en los sistemas distribuidos que en los sistemas centralizados, la capacidad que tienen muchos sistemas de continuar trabajando a pesar del fallo en uno de los sitios produce una mayor disponibilidad. La disponibilidad es crucial para los sistemas de bases de datos que se utilizan en aplicaciones de tiempo real. Que, por ejemplo, una línea aérea pierda el acceso a los datos puede provocar la pérdida de potenciales compradores de billetes en favor de la competencia.

18.4.1. Un ejemplo de una base de datos distribuida

Considérese un sistema bancario compuesto por cuatro sucursales situadas en cuatro ciudades diferentes. Cada sucursal posee su propia computadora con una base de datos que alberga todas las cuentas abiertas en dicha sucursal. Así, cada una de estas instalaciones se considera un sitio. También hay un único sitio que mantiene la información relativa a todas las sucursales del banco. Cada sucursal dispone (entre otras) de una relación *cuenta*(*Esquema-cuenta*), donde

Esquema-cuenta = (*número-cuenta, nombre-sucursal, saldo*)

El sitio que contiene información acerca de las cuatro sucursales mantiene la relación *sucursal*(*Esquema-sucursal*), donde

Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activos*)

Existen otras relaciones en los distintos sitios que serán ignoradas para los propósitos del ejemplo.

Para ilustrar la diferencia entre los dos tipos de transacciones considérese la transacción que suma 50 € a la cuenta C-177 situada en la sucursal de Cercedilla. La transacción se considera local si ésta comenzó en la sucursal de Cercedilla; en otro caso, se considera global. Una transacción que transfiere 50 € desde la cuenta C-177 a la cuenta C-305, que se encuentra en la sucursal de Guadarrama, es una transacción global, ya que como resultado de su ejecución se accede a datos de dos sitios diferentes.

En un sistema distribuido de bases de datos ideal, los sitios deberían compartir un esquema global común (aunque algunas relaciones se puedan almacenar sólo en algunos sitios), todos los sitios deberían ejecutar el mismo software de gestión de bases de datos distribuidas, y los sitios deberían conocer la existencia de los

demás. Si una base de datos distribuida se construye partiendo de cero, realmente debería ser posible lograr los objetivos anteriores. Sin embargo, en la realidad, una base de datos distribuida se tiene que construir enlazando múltiples sistemas de bases de datos que ya existen, cada uno con su propio esquema y posiblemente ejecutando diferente software de gestión de bases de datos. A veces, tales sistemas reciben el nombre de **sistemas de bases de datos múltiples** o **sistemas distribuidos y heterogéneos de bases de datos**. En el Apartado 19.8 se discuten dichos sistemas y se muestra cómo conseguir un cierto grado de control global a pesar de la heterogeneidad de los sistemas que lo componen.

18.4.2. Aspectos de la implementación

La atomicidad de las transacciones es un aspecto importante de la construcción de un sistema distribuido de bases de datos. Si una transacción se ejecuta a lo largo de dos sitios, a menos que los diseñadores del sistema sean cuidadosos, puede comprometerse en un sitio y cancelarse en otro, lo que conduciría a un estado de inconsistencia. Los protocolos de compromiso de transacciones aseguran que tales situaciones no se produzcan. El *protocolo de compromiso de dos fases* (C2F) es el más utilizado de estos protocolos.

La idea básica del C2F es que cada sitio ejecuta la transacción justo hasta antes del compromiso, y entonces deja la decisión del compromiso a un único sitio coordinador; se dice que en ese punto la transacción está en estado *preparada* en el sitio. El coordinador decide comprometer la transacción sólo si la transacción alcanza el estado preparada en cada sitio donde se ejecutó; en otro caso (por ejemplo, si la transacción se canceló en algún sitio), el coordinador decide cancelar la transacción. Todos los sitios donde la transacción se ejecutó deben acatar la decisión del coordinador. Si un sitio falla cuando una transacción se encuentra en estado preparada, cuando el sitio se recupere del fallo debería estar en posición de comprometer o cancelar la transacción dependiendo de la decisión del coordinador. El protocolo C2F se describe en detalle en el Apartado 19.4.1.

El control de concurrencia es otra característica de una base de datos distribuida. Como una transacción puede acceder a elementos de datos de varios sitios, los administradores de transacciones de varios sitios pueden necesitar coordinarse para implementar el control de la concurrencia. Si se utiliza bloqueo (como casi siempre sucede en la práctica), el bloqueo se puede realizar de forma local en los sitios que contienen los elementos de datos accedidos, pero también existe posibilidad de un interbloqueo que involucre a transacciones originadas en múltiples sitios. Por lo tanto, es necesario llevar la detección de interbloqueos a lo largo de múltiples sitios. Los fallos son más comunes en los sistemas distribuidos, dado que no sólo las computadoras pueden fallar, sino que también pueden fallar los enlaces de comunicaciones. La réplica de los elementos de datos, que es la clave para el funcionamiento continuado de las bases de datos distribuidas cuando ocurren fallos, complica aún más el control de la concurrencia. El Apartado 19.5 proporciona más detalles sobre el control de la concurrencia en bases de datos distribuidas.

Los modelos estándar de transacciones, basados en múltiples acciones llevadas a cabo por una única unidad de programa, son a menudo inapropiadas para realizar tareas que cruzan los límites de las bases de datos que no pueden o no cooperarán para implementar protocolos como el C2F. Para estas tareas se utilizan generalmente técnicas alternativas, basadas en *mensajería persistente* para las comunicaciones.

Cuando las tareas a realizar son complejas, involucrando múltiples bases de datos y múltiples interacciones con humanos, la coordinación de las tareas y asegurar las propiedades de las transacciones para las tareas se vuelve más complicado. Los *sistemas de gestión de flujos de trabajo* son sistemas diseñados para ayudar en la realización de dichas tareas. El Apartado 19.4.3 describe la mensajería persistente, mientras que el Apartado 24.2 describe los sistemas de gestión de flujos de trabajo.

En caso de que una empresa tenga que escoger entre una arquitectura distribuida y una arquitectura centralizada para implementar una aplicación, el arquitecto del sistema debe sopesar las ventajas frente a las desventajas de la distribución de datos. Ya se han examinado las ventajas de utilizar bases de datos distribuidas. El principal inconveniente de los sistemas distribuidos de bases de datos es la complejidad añadida que es necesaria para garantizar la coordinación apropiada entre los sitios. Esta creciente complejidad tiene varias facetas:

- **Coste de desarrollo del software.** La implementación de un sistema distribuido de bases de datos es más difícil y, por lo tanto, más costoso.
- **Mayor probabilidad de errores.** Como los sitios que constituyen el sistema distribuido operan en paralelo es más difícil asegurarse de la corrección de los algoritmos, del funcionamiento especial durante los fallos de parte del sistema así como de la recuperación. Son probables errores extremadamente sutiles.
- **Mayor sobrecarga de procesamiento.** El intercambio de mensajes y el cómputo adicional necesario para conseguir la coordinación entre los distintos sitios constituyen una forma de sobrecarga que no surge en los sistemas centralizados.

Existen varios enfoques acerca del diseño de las bases de datos distribuidas que abarcan desde los diseños completamente distribuidos hasta los que incluyen un alto grado de centralización. Se estudiarán en el Capítulo 19.

18.5. TIPOS DE REDES

Las bases de datos distribuidas y los sistemas cliente-servidor se construyen en torno a las redes de comunicación. Existen básicamente dos clases de redes: las **redes de área local** y las **redes de área amplia**. La diferencia principal entre ambas es la forma en que están distribuidas geográficamente. Las redes de área local están compuestas por procesadores distribuidos en áreas geográficas pequeñas tales como un único edificio o varios edificios adyacentes. Por su parte, las redes de área amplia se componen de un número determinado de procesadores autónomos que están distribuidos a lo largo de una extensa área geográfica (como puede ser España o el mundo entero). Estas diferencias implican importantes variaciones en la velocidad y en la fiabilidad de la red de comunicación y quedan reflejadas en el diseño del sistema operativo distribuido.

18.5.1. Redes de área local

Las **redes de área local** (**LANs, Local Area Networks**) (Figura 18.10) surgen a principios de los 70 como una forma de comunicación y de compartimiento de datos entre varias computadoras. La gente se dio cuenta de que en muchas empresas era más económico tener muchas computadoras pequeñas, cada una de ellas con sus propias aplicaciones, que un enorme y único sistema. La conexión de estas pequeñas computadoras formando una red parece un paso natural porque, probablemente, cada pequeña computadora necesite acceder a un conjunto complementario de dispositivos periféricos (como discos e impresoras) y porque en una empresa suele ser necesaria el compartimiento de algunos datos.

Las LAN se utilizan generalmente en un entorno de oficina. Todos los puestos de estos sistemas están próximos entre sí por lo que los enlaces de comunicación suelen poseer una mayor velocidad y una tasa de errores más baja que la que se da en las redes de área amplia. Los enlaces más comunes en una red de área local son el par trenzado, el cable coaxial, la fibra óptica y, cada vez más, las conexiones inalámbricas. La velocidad de comunicación varía entre unos pocos megabits por segundo (en las redes de área local inalámbricas), a un gigabit por segundo para Gigabit Ethernet. La Ethernet estándar funciona a 10 megabits por segundo, mientras que Fast Ethernet llega a 100 megabits por segundo.

Una **red de área de almacenamiento (SAN, Storage-Area Network)** es un tipo especial de red de área local de alta velocidad destinada a conectar numerosos bancos de dispositivos de almacenamiento (discos) a las computadoras que utilizan los datos. Así, las redes de área de almacenamiento ayudan a construir *sistemas de discos compartidos* a gran escala. El motivo para utilizar redes de área de almacenamiento para conectar múltiples computadoras a grandes bancos de dispositivos de almacenamiento es esencialmente el mismo que para las bases de datos de disco compartido, a saber:

- Escalabilidad añadiendo más computadoras.

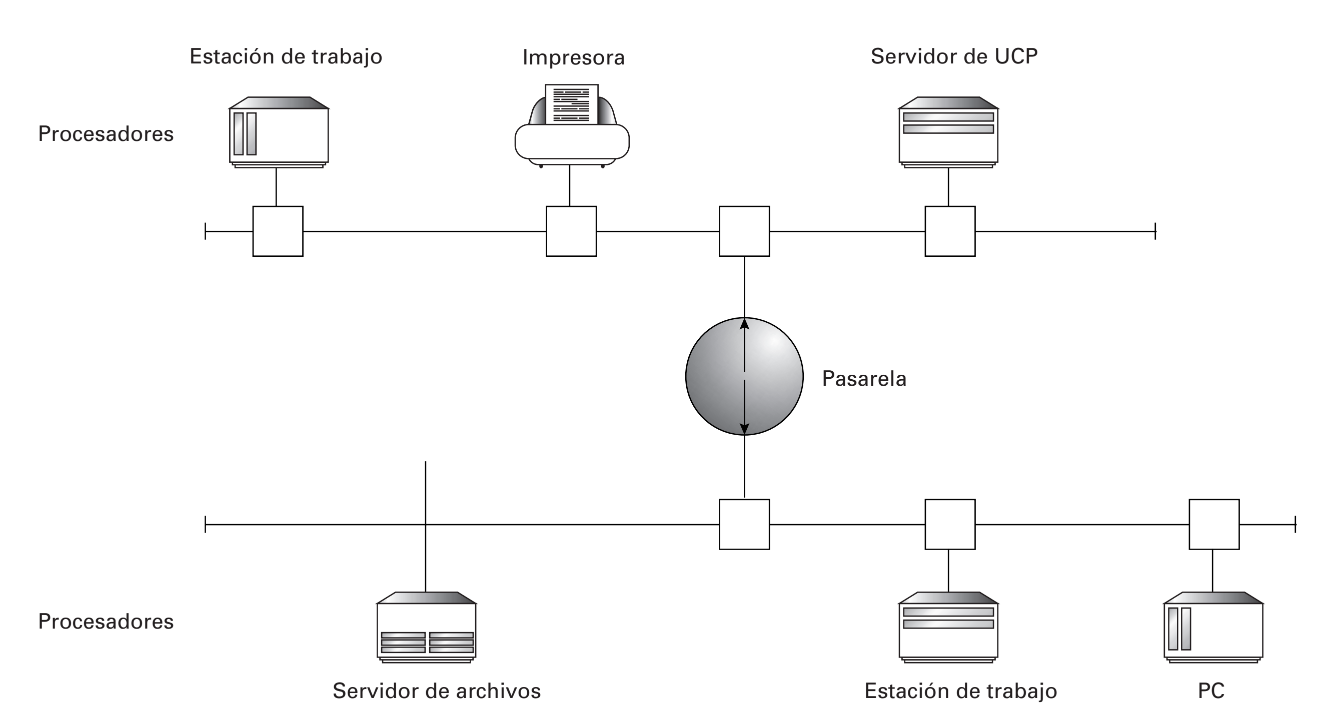


FIGURA 18.10. Red de área local.

- Alta disponibilidad, ya que los datos están todavía accesibles incluso si una computadora falla.

Las organizaciones RAID se utilizan en dispositivos de almacenamiento para asegurar la alta disponibilidad de los datos, permitiendo continuar al procesamiento incluso si discos individuales fallan. Las redes de área de almacenamiento se construyen normalmente con redundancia, como múltiples caminos entre nodos, así si un componente como un enlace o una conexión a la red falla, la red continúa funcionando.

18.5.2. Redes de área amplia

Las **redes de área amplia** (**WAN, Wide Area Networks**) surgen a finales de los 60 principalmente como un proyecto de investigación académica para proporcionar una comunicación eficiente entre varios lugares permitiendo que una gran comunidad de usuarios pudiera compartir hardware y software de una manera conveniente y económica. A principios de los años 60 se desarrollaron sistemas que permitían que terminales remotos se conectaran a una computadora central a través de la línea telefónica, pero no eran verdaderas WANs. *Arpanet* fue la primera WAN que se diseñó y se desarrolló. El trabajo en Arpanet comenzó en 1968. Arpanet ha crecido de tal forma que ha pasado de ser una red experimental de cuatro puestos a una red de redes extendida por todo el mundo, **Internet**, abarcando a cientos de millones de sistemas de computación. Los enlaces típicos de Internet son las líneas de fibra óptica y, a veces, los canales vía satélite. Las transferencias de datos para los enlaces de área amplia varían normalmente entre los pocos megabits por segundo y los cientos de gigabits por segundo. El último enlace, hasta el puesto del usuario, se basa a menudo en la tecnología de *bucle de suscriptor digital* (DSL, digital subscriber loop) (que soporta unos pocos megabits por segundo), o módem de cable (que soporta 10 megabits por segundo) o conexiones de módem telefónico sobre líneas telefónicas (que soportan hasta 56 kilobits por segundo).

Pueden distinguirse dos tipos de WANs:

- En las WAN de **conexión discontinua**, como las basadas en conexiones por radio, las computadoras están conectadas a la red sólo durante intervalos del tiempo.
- En las WAN de **conexión continua**, como Internet, las computadoras están conectadas a la red continuamente.

Las redes que no están continuamente conectadas no suelen permitir las transacciones entre distintos sitios, pero pueden almacenar copias locales de los datos remotos y actualizarlas periódicamente (por ejemplo, todas las noches). Para las aplicaciones en las que la consistencia no es un factor crítico, como ocurre en el compartimiento de documentos, los sistemas de software de grupo como Lotus Notes permiten realizar localmente las actualizaciones de los datos remotos y propagar más tarde dichas actualizaciones al sitio remoto. Debe detectarse y resolverse el riesgo potencial de conflicto entre varias actualizaciones realizadas en sitios diferentes. Más adelante, en el Apartado 23.5.4, se describe un mecanismo para detectar actualizaciones conflictivas; el mecanismo de resolución de actualizaciones conflictivas es, sin embargo, dependiente de la aplicación.

18.6. RESUMEN

- Los sistemas centralizados de bases de datos se ejecutan completamente en una única computadora. Con el crecimiento de las computadoras personales y las redes de área local, se ha ido desplazando hacia el lado del cliente la funcionalidad de la parte visible al usuario de la base de datos de modo que los sistemas servidores provean la funcionalidad del sistema subyacente. Los protocolos de interfaz cliente-servidor han ayudado al crecimiento de los sistemas de bases de datos cliente-servidor.
- Los servidores pueden ser servidores de transacciones o servidores de datos, aunque el uso de los servidores de transacciones excede ampliamente el uso de los servidores de datos para proporcionar servicios de bases de datos.
  - Los servidores de transacciones tienen múltiples procesos, ejecutándose posiblemente en múltiples procesadores. Dado que estos procesos tienen acceso a los datos comunes, como la memoria intermedia de la base de datos, los sistemas almacenan dichos datos en memoria compartida. Además de los procesos que gestionan consultas, hay procesos del sistema que realizan tareas como la gestión de los bloqueos y del registro y los puntos de revisión.
  - Los sistemas servidores de datos suministran datos sin formato a los clientes. Tales sistemas se esfuerzan en minimizar la comunicación entre clientes y servidores usando caché de datos y de bloqueos en los clientes. Los sistemas paralelos de bases de datos utilizan optimizaciones similares.
- Los sistemas paralelos de bases de datos consisten en varios procesadores y varios discos conectados a través de una red de interconexión de alta velocidad. La ganancia de velocidad mide cuánto puede incrementarse la velocidad de procesamiento al incrementarse el paralelismo dada una transacción. La ampliabili-

dad mide lo bien que se gestiona un mayor número de transacciones cuando se incrementa el paralelismo. La interferencia, el sesgo y los costes de inicio actúan como barreras para obtener la ganancia de velocidad y la ampliabilidad ideales.

- Las arquitecturas paralelas de bases de datos pueden clasificarse en arquitecturas de memoria compartida, de disco compartido, sin compartimiento o jerárquicas. Estas arquitecturas tienen distintos compromisos entre la ampliabilidad y la velocidad de comunicación.
- Una base de datos distribuida es un conjunto de bases de datos parcialmente independientes que (idealmente) comparten un esquema común y coordinan el procesamiento de transacciones que acceden a datos remotos. Los procesadores se comunican entre sí a través de una red de comunicación que gestiona el encaminamiento y las estrategias de conexión.
- Hay dos tipos principales de redes de comunicación: las redes de área local y las de área amplia. Las redes de área local conectan nodos que están distribuidos sobre áreas geográficas pequeñas tales como un único edificio o varios edificios adyacentes. Las redes de área amplia conectan nodos a lo largo de una extensa área geográfica. Hoy en día, la red de área amplia más extensa que se utiliza es Internet.

  Las redes de área de almacenamiento son un tipo especial de redes de área local diseñadas para proporcionar interconexión rápida entre grandes bancos de dispositivos de almacenamiento y múltiples computadoras.

TÉRMINOS DE REPASO

- Ampliabilidad
  - Ampliabilidad de transacciones
  - Ampliabilidad lineal
  - Ampliabilidad por lotes
  - Ampliabilidad sublineal
- Arquitectura de memoria no uniforme
- Arquitecturas paralelas de bases de datos
  - Disco compartido (agrupaciones)
  - Jerárquico
  - Memoria compartida
  - Sin compartimiento
- Bases de datos distribuidas
  - Autonomía local
  - Sitios (nodos)
  - Transacción global
  - Transacción local
- Costes de inicio
- Estructura de proceso de la base de datos
- Exclusión mutua
- Ganancia de velocidad
  - Ganancia de velocidad lineal
  - Ganancia de velocidad sublineal
- Hebra
- Interferencia
- Memoria virtual distribuida
- Paralelismo de grano fino
- Paralelismo de grano grueso
- Procesos servidor
  - Proceso administrador de bloqueos
  - Proceso escritor de bases de datos
  - Proceso escritor del registro
  - Proceso monitor de procesos
  - Proceso punto de revisión
- Productividad
- Redes de interconexión
  - Bus
  - Hipercubo
  - Malla
- Sesgo
- Servidor de consultas
- Servidor de datos
  - Caché de bloqueos
  - Caché de datos
  - Coherencia de caché
  - Comunicar
  - Liberación de bloqueos
  - Preextracción
- Servidor de transacciones
- Sistema con múltiples bases de datos
- Sistemas centralizados
- Sistemas cliente-servidor
- Sistemas distribuidos
- Sistemas paralelos
- Sistemas servidores
- Tiempo de respuesta
- Tipos de redes
  - Redes de área de almacenamiento (SAN)
  - Redes de área amplia (WAN)
  - Redes de área local (LAN)
- Tolerancia ante fallos

EJERCICIOS

18.1. ¿Por qué es relativamente fácil trasladar una base de datos desde una máquina con un único procesador a otra con varios procesadores si no es necesario hacer paralelas las consultas individuales?

18.2. Las arquitecturas servidoras de transacciones son populares entre las bases de datos relacionales cliente-servidor, en las que las transacciones son cortas. Por el contrario, las arquitecturas servidoras de datos son populares entre los sistemas cliente-servidor de bases de datos orientadas a objetos, donde las transacciones son relativamente largas. Dense dos razones por las que los servidores de datos puedan ser populares entre las bases de datos orientadas a objetos y no los sean entre las bases de datos relacionales.

18.3. En lugar de almacenar estructuras compartidas en memoria compartida, una arquitectura alternativa podría ser almacenarlas en la memoria local de un proceso especial, y acceder a los datos compartidos mediante comunicación entre procesos con el proceso. ¿Cuál sería la desventaja de dicha arquitectura?

18.4. La máquina que hace de servidor en los sistemas cliente-servidor típicos es mucho más potente que los clientes, es decir, su procesador es más rápido, puede tener varios procesadores, tiene más memoria y tiene discos de mayor capacidad. En vez de esto, considérese el caso en el que los clientes y el servidor tuvieran exactamente la misma potencia. ¿Tendría sentido construir un sistema cliente-servidor en ese caso? ¿Por qué? ¿Qué caso se ajustaría mejor a una arquitectura servidora de datos?

18.5. Considérese un sistema de base de datos orientada a objetos sobre una arquitectura cliente-servidor en la que el servidor actúa como servidor de datos.

a. ¿Cuál es el efecto de la velocidad de interconexión entre el cliente y el servidor en los casos de envío de páginas y de objetos?

b. Si se utiliza envío de páginas, la caché de datos en el cliente puede organizarse como una caché de objetos o una caché de páginas. La caché de páginas almacena los datos en unidades de páginas mientras que la caché de objetos almacena los datos en unidades de objetos. Supóngase que los objetos son más pequeños que una página. Descríbase una ventaja de la caché de objetos frente a la caché de páginas.

18.6. ¿Qué es la liberación de bloqueos y bajo qué condiciones es necesaria? ¿Por qué no es necesario si la unidad de envío de datos es un elemento?

18.7. Suponga que se encuentra a cargo de las operaciones de la base de datos de una empresa cuyo trabajo principal es el de procesar transacciones. Suponga que la empresa crece rápidamente cada año y que el sistema informático actual se ha quedado pequeño. Cuando escoja una nueva computadora paralela, ¿qué factor será más importante: la ganancia de velocidad, la ampliabilidad por lotes o la ampliabilidad de transacciones? ¿Por qué?

18.8. Supóngase una transacción escrita en C con código SQL incorporado que ocupa el 80 por ciento del tiempo en ejecutar el código SQL y sólo el 20 por ciento restante en el código C. ¿Qué ganancia de velocidad puede esperarse si sólo se hace paralelo el código SQL? Justifíquese la respuesta.

18.9. En un sistema de procesamiento de transacciones, ¿cuáles son los factores que trabajan en contra de la ampliabilidad lineal? ¿Cuál de esos factores es probablemente el más importante en cada una de estas arquitecturas: memoria compartida, disco compartido y sin compartimiento?

18.10. Considérese un banco que dispone de un conjunto de sitios en los que se ejecuta un sistema de base de datos. Supóngase que la transferencia electrónica de dinero entre ellos es el único modo de interacción de las bases de datos. ¿Puede un sistema tal ser calificado como distribuido? ¿Por qué?

18.11. Considérese una red basada en líneas de acceso telefónico en la que los sitios se comunican periódicamente, por ejemplo, todas las noches. Estas redes suelen tener un servidor y varios clientes. Los sitios que actúan como clientes están conectados sólo con el servidor e intercambian los datos con el resto de clientes almacenándolos en el servidor y recuperando los almacenados por otros clientes en el servidor. ¿Cuál es la ventaja de tal arquitectura frente a una en la que un sitio pueda intercambiar datos con otro mediante acceso telefónico directo?

NOTAS BIBLIOGRÁFICAS

Los libros de Patterson y Hennessy [1995] y Stone [1993] proporcionan una buena introducción al área de la arquitectura de computadoras.

Gray y Reuter [1993] dan una descripción en su libro del procesamiento de transacciones incluyendo la arquitectura de cliente-servidor y los sistemas distribuidos. Geiger [1995] y Signore et al. [1995] describen la norma ODBC para la conectividad cliente-servidor. En North [1995] pueden encontrarse descripciones de varias herramientas para el acceso cliente-servidor a bases de datos.

Carey et al. [1991] y Franklin et al. [1993] describen técnicas de caché de datos para los sistemas de bases de datos cliente-servidor. Biliris y Orenstein [1994] tratan los sistemas de gestión de almacenamiento de obje-

tos incluyendo aspectos relacionados con los sistemas cliente-servidor. En Franklin et al. [1992] y Mohan y Narang [1994] pueden encontrarse algunas técnicas para la recuperación en sistemas cliente-servidor.

De Witt y Gray [1992] describen los sistemas paralelos de bases de datos incluyendo sus propias arquitecturas y medidas de rendimiento. Duncan [1990] introduce una vista de las arquitecturas paralelas de computadoras. Dubois y Thakkar [1992] es una colección de artículos sobre las arquitecturas ampliables de memoria compartida.

Ozsu y Valduriez [1999], Bell y Grimson [1992] y Ceri y Pelagatti [1984] proporcionan textos sobre los sistemas distribuidos de bases de datos. Se pueden encontrar más referencias sobre los sistemas de bases de datos paralelos y distribuidos en las notas bibliográficas de los Capítulos 20 y 19, respectivamente.

Comer y Droms [1999] y Thomas [1996] describen las redes de computadoras e Internet. Tanenbaum [1996] y Halsall [1992] proporcionan revisiones generales de las redes de computadoras. Prycker [1993] examina las Redes MTA y los conmutadores.

CAPÍTULO

19 BASES DE DATOS DISTRIBUIDAS

A diferencia de los sistemas paralelos, en los que los procesadores se hallan estrechamente acoplados y constituyen un único sistema de bases de datos, los sistemas distribuidos de bases de datos consisten en sitios débilmente acoplados que no comparten ningún componente físico. Además, puede que los sistemas de bases de datos que se ejecutan en cada sitio tengan un grado sustancial de independencia mutua. La estructura básica de los sistemas distribuidos se discutió en el Capítulo 18.

Cada sitio puede participar en la ejecución de transacciones que tienen acceso a los datos de uno o varios de los sitios. La diferencia principal entre los sistemas de bases de datos centralizados y los distribuidos es que, en los primeros, los datos residen en una única ubicación, mientras que en los segundos los datos residen en varias ubicaciones. La distribución de los datos es causa de muchas dificultades en el procesamiento de las transacciones y de las consultas. En este capítulo se abordarán esas dificultades.

Se comienza por clasificar las bases de datos distribuidas en homogéneas y heterogéneas en el Apartado 19.1. Luego se aborda el problema del almacenamiento de los datos en las bases de datos distribuidas en el Apartado 19.2. En el Apartado 19.3 se esboza un modelo de procesamiento de las transacciones en bases de datos distribuidas. En el Apartado 19.4 se describe el modo de implementar transacciones atómicas en bases de datos distribuidas mediante protocolos de compromiso especiales. En el Apartado 19.5 se describe el control de la concurrencia en las bases de datos distribuidas. En el Apartado 19.6 se esboza el modo de proporcionar una elevada disponibilidad en bases de datos distribuidas aprovechando las réplicas, de modo que el sistema pueda continuar procesando las transacciones aunque se produzca un fallo. El procesamiento de las consultas en las bases de datos distribuidas se aborda en el Apartado 19.7. En el Apartado 19.8 se esbozan aspectos del manejo de bases de datos heterogéneas. En el Apartado 19.9 se describen los sistemas de directorio, que pueden considerarse una forma especializada de bases de datos distribuidas.

19.1. BASES DE DATOS HOMOGÉNEAS Y HETEROGÉNEAS

En las **bases de datos distribuidas homogéneas** todos los sitios tienen idéntico software de sistemas gestores de bases de datos, son conscientes de la existencia de los demás sitios y acuerdan cooperar en el procesamiento de las solicitudes de los usuarios. En estos sistemas los sitios locales renuncian a una parte de su autonomía en cuanto a su derecho a modificar los esquemas o el software del sistema gestor de bases de datos. Ese software también debe cooperar con los demás sitios en el intercambio de la información sobre las transacciones para hacer posible el procesamiento de las transacciones entre varios sitios.

A diferencia de lo anterior, en las **bases de datos distribuidas heterogéneas** sitios diferentes puede que utilicen esquemas diferentes y diferente software de gestión de sistemas de bases de datos. Puede que unos sitios no sean conscientes de la existencia de los demás y puede que sólo proporcionen facilidades limitadas para la cooperación en el procesamiento de las transacciones. Las diferencias en los esquemas suelen constituir un problema importante para el procesamiento de las consultas, mientras que la divergencia del software supone un inconveniente para el procesamiento de transacciones que tengan acceso a varios sitios.

Este capítulo se centrará en las bases de datos distribuidas homogéneas. No obstante, en el Apartado 19.8 se discutirán brevemente los aspectos del procesamiento de las consultas en los sistemas de bases de datos distribuidas heterogéneas. Los aspectos del procesamiento de las transacciones en dichos sistemas se tratan más adelante, en el Apartado 24.6.

19.2. ALMACENAMIENTO DISTRIBUIDO DE DATOS

Considérese una relación *r* que hay que almacenar en la base de datos. Hay dos enfoques del almacenamiento de esta relación en la base de datos distribuida:

- **Réplica.** El sistema conserva réplicas (copias) idénticas de la relación y guarda cada réplica en un sitio diferente. La alternativa a las réplicas es almacenar sólo una copia de la relación *r*.
- **Fragmentación.** El sistema divide la relación en varios fragmentos y guarda cada fragmento en un sitio diferente.

La fragmentación y la réplica pueden combinarse: Las relaciones pueden dividirse en varios fragmentos y puede haber varias réplicas de cada fragmento. En los subapartados siguientes se profundizará en cada una de estas técnicas.

19.2.1. Réplica de datos

Si la relación *r* se replica, se guarda una copia de dicha relación en dos o más sitios. En el caso más extremo se tiene una **réplica completa**, en la que se guarda una copia en cada sitio del sistema.

Hay varias ventajas y desventajas en las réplicas.

- **Disponibilidad.** Si alguno de los sitios que contiene la relación *r* falla, la relación puede hallarse en otro sitio distinto. Por tanto, el sistema puede seguir procesando las consultas que impliquen a *r*, pese al fallo del sitio.
- **Paralelismo incrementado.** En caso de que la mayoría de los accesos a la relación *r* sólo resulten en la lectura de la relación, varios sitios pueden procesar en paralelo las lecturas que impliquen a *r*. Cuantas más réplicas de *r* haya, mayor será la posibilidad de que los datos necesarios se hallen en el sitio en que se ejecuta la transacción. Por tanto, la réplica de los datos minimiza el movimiento de los datos entre los sitios.
- **Sobrecarga incrementada durante la actualización.** El sistema debe asegurar que todas las réplicas de la relación *r* sean consistentes; en caso contrario pueden producirse cómputos erróneos. Por eso, siempre que se actualiza *r*, hay que propagar la actualización a todos los sitios que contienen réplicas. El resultado es una sobrecarga incrementada. Por ejemplo, en un sistema bancario, en el que se replica en varios sitios la información de las cuentas, es necesario asegurarse de que el saldo de cada cuenta concuerde en todos los sitios.

En general, la réplica mejora el rendimiento de las operaciones leer y aumenta la disponibilidad de los datos para las transacciones sólo de lectura. No obstante, las transacciones de actualización suponen una mayor sobrecarga. El control de las actualizaciones de actualización realizadas por varias transacciones en los datos replicados resulta más complicado que en los sistemas centralizados, que se vieron en el Capítulo 16. Se puede simplificar la gestión de las réplicas de la relación *r* escogiendo una de ellas como **copia principal** de *r*. Por ejemplo, en un sistema bancario, las cuentas pueden asociarse con el sitio en que se abrieron. De manera parecida, en un sistema de reserva de billetes de avión, los vuelos pueden asociarse con el sitio en que se origina el vuelo. Se examinará el esquema de copias principales y otras opciones del control de la concurrencia distribuida en el Apartado 19.5.

19.2.2. Fragmentación de los datos

Si la relación *r* se fragmenta, *r* se divide en varios *fragmentos* $r_1, r_2, \dots, r_n$. Estos fragmentos contienen suficiente información como para permitir la reconstrucción de la relación original *r*. Hay dos esquemas diferentes de fragmentación de las relaciones: fragmentación *horizontal* y fragmentación *vertical*. La fragmentación horizontal divide la relación asignando cada tupla de *r* en uno o más fragmentos. La fragmentación vertical divide la relación descomponiendo el esquema *R* de la relación *r*.

Estos enfoques se ilustrarán fragmentando la relación *cuenta*, con el esquema

esquema-cuenta = (*número-cuenta*, *nombre-sucursal*, *saldo*)

En la **fragmentación horizontal** la relación *r* se divide en varios subconjuntos, $r_1, r_2, \dots, r_n$. Cada tupla de la relación *r* debe pertenecer como mínimo a uno de los fragmentos, de modo que se pueda reconstruir la relación original, si fuera necesario.

A modo de ejemplo, la relación *cuenta* puede dividirse en varios fragmentos, cada uno de los cuales consiste en tuplas de cuentas que pertenecen a una sucursal concreta. Si el sistema bancario sólo tiene dos sucursales (Guadarrama y Cercedilla) habrá dos fragmentos diferentes:

$$cuenta_1 = \sigma_{nombre\text{-}sucursal\ =\ \text{«Guadarrama»}}\ (cuenta)$$
$$cuenta_2 = \sigma_{nombre\text{-}sucursal\ =\ \text{«Cercedilla»}}\ (cuenta)$$

La fragmentación horizontal suele utilizarse para conservar las tuplas en los sitios en que más se utilizan, para minimizar la transferencia de datos.

En general, los fragmentos horizontales pueden definirse como una *selección* de la relación global *r*. Es decir, se utiliza un predicado P_i para construir fragmentos r_i:

$$r_i = \sigma_{P_i}\ (r)$$

Se reconstruye la relación r tomando la unión de todos los fragmentos; es decir,

$$r = r_1 \cup r_2 \cup \cdots \cup r_n$$

En el ejemplo los fragmentos son disjuntos. Al cambiar los predicados de selección empleados para crear los fragmentos se puede hacer que una tupla concreta de r aparezca en más de uno de los fragmentos r_i.

En su forma más sencilla la fragmentación vertical es igual que la descomposición (véase el Capítulo 7). La **fragmentación vertical** de $r(R)$ implica la definición de varios subconjuntos de atributos $R_1, R_2, \ldots, R_n$ del esquema R de modo que

$$R = R_1 \cup R_2 \cup \cdots \cup R_n$$

Cada fragmento r_i de r se define mediante

$$r_i = \Pi_{R_i}(r)$$

La fragmentación debe hacerse de modo que se pueda reconstruir la relación r a partir de los fragmentos tomando la reunión natural

$$r = r_1 \bowtie r_2 \bowtie r_3 \bowtie \cdots \bowtie r_n$$

Una manera de asegurar que la relación r pueda reconstruirse es incluir los atributos de la clave principal de R en cada uno de los fragmentos R_i. De manera más general, se puede utilizar cualquier superclave. Suele resultar conveniente añadir un atributo especial, denominado *id-tupla*, al esquema R. El valor id-tupla de una tupla es un valor único que distingue cada tupla de todas las demás. El atributo id-tupla, por tanto, sirve como clave candidata para el esquema aumentado y se incluye en cada uno de los fragmentos R_i. La dirección física o lógica de la tupla puede utilizarse como id-tupla, dado que cada tupla tiene una dirección única.

Para ilustrar la fragmentación vertical considérese una base de datos universitaria con una relación *info-empleado* que almacena, para cada empleado, *id-empleado*, *nombre*, *puesto* y *salario*. Por motivos de preservación de la intimidad puede que esta relación se fragmente en una relación *empleado-infoprivada* que contenga *id-empleado* y *salario*, y en otra relación *empleado-infopública* que contenga los atributos *id-empleado*, *nombre* y *puesto*. Puede que las dos relaciones se almacenen en sitios diferentes, nuevamente, por motivos de seguridad.

Se pueden aplicar los dos tipos de fragmentación a un solo esquema; por ejemplo, los fragmentos obtenidos de la fragmentación horizontal de una relación pueden dividirse nuevamente de manera vertical. Los fragmentos también pueden replicarse. En general, los fragmentos pueden replicarse, las réplicas de los fragmentos pueden fragmentarse más, etcétera, etcétera.

19.2.3. Transparencia

No se debe exigir a los usuarios de los sistemas distribuidos de bases de datos que conozcan la ubicación física de los datos ni el modo en que se puede tener acceso a ellos en un sitio local concreto. Esta característica, denominada **transparencia de los datos**, puede adoptar varias formas:

- **Transparencia de la fragmentación.** No se exige a los usuarios que conozcan el modo en que se ha fragmentado la relación.
- **Transparencia de la réplica.** Los usuarios ven cada objeto de datos como lógicamente único. Puede que el sistema distribuido replique los objetos para incrementar el rendimiento del sistema o la disponibilidad de los datos. Los usuarios no deben preocuparse por los objetos que se hayan replicado ni por la ubicación de esas réplicas.
- **Transparencia de la ubicación.** No se exige a los usuarios que conozcan la ubicación física de los datos. El sistema distribuido de bases de datos debe poder hallar los datos siempre que la transacción del usuario facilite el identificador de los datos.

Los elementos de datos (como las relaciones, los fragmentos y las réplicas) deben tener nombres únicos. Esta propiedad es fácil de asegurar en una base de datos centralizada. En las bases de datos distribuidas, sin embargo, hay que tener cuidado para asegurarse de que dos sitios no utilicen el mismo nombre para elementos de datos diferentes.

Una solución a este problema es exigir que todos los nombres se registren en un **servidor de nombres** central. El servidor de nombres ayuda a asegurar que el mismo nombre no se utilice para elementos de datos diferentes. También se puede utilizar el servidor de nombres para ubicar un elemento de datos, dado el nombre del elemento. Este enfoque, sin embargo, presenta dos inconvenientes principales. En primer lugar, puede que el servidor de nombres se transforme en un cuello de botella para el rendimiento cuando los elementos de datos se ubican por sus nombres, lo que da lugar a un bajo rendimiento. En segundo lugar, si el servidor de nombres queda fuera de servicio, puede que no sea posible que siga funcionando ningún otro sitio del sistema distribuido.

Un enfoque alternativo más utilizado exige que cada sitio anteponga su propio identificador de sitio a cualquier nombre que genere. Este enfoque asegura que dos sitios no generen nunca el mismo nombre (dado que cada sitio tiene un identificador único). Además, no se necesita ningún control centralizado. Esta solución, no obstante, no logra conseguir la transparencia de la ubicación, dado que se adjuntan a los nombres los identificadores de los sitios. Así, se puede hacer referencia a la relación *cuenta* como *sitio*17.*cuenta*, o *cuenta*@*sitio*17, en lugar de meramente *cuenta*. Muchos sistemas de bases de datos utilizan la dirección de internet de los sitios para identificarlos.

Para superar este problema el sistema de bases de datos puede crear un conjunto de nombres alternativos o **alias** para los elementos de datos. Así, los usuarios se pueden referir a los artículos de datos mediante nombres sencillos que el sistema traduce en los nombres completos. La asignación de los alias a los nombres reales puede almacenarse en cada sitio. Con los alias el usuario puede ignorar la ubicación física de los elementos de datos. Además, el usuario no se verá afectado si el administrador de la base de datos decide trasladar un elemento de datos de un sitio a otro.

Los usuarios no deben tener que hacer referencia a una réplica concreta de un elemento de datos. En vez de eso, el sistema debe determinar la réplica a la que hay que hacer referencia en las solicitudes leer, y actualizar todas las réplicas en las solicitudes escribir. Se puede asegurar que lo haga manteniendo una tabla de catálogo, que el sistema utiliza para determinar todas las réplicas del elemento de datos.

19.3. TRANSACCIONES DISTRIBUIDAS

El acceso a los diferentes elementos de datos en los sistemas distribuidos suele realizarse mediante transacciones, que deben preservar las propiedades ACID (Apartado 15.1). Hay dos tipos de transacciones que se deben considerar. Las **transacciones locales** son las que tienen acceso a los datos y los actualizan sólo en una base de datos local; las **transacciones globales** son las que tienen acceso a datos y los actualizan en varias bases de datos locales. Se pueden asegurar las propiedades ACID de las transacciones locales como se describe en los capítulos 15, 16 y 17. Sin embargo, para las transacciones globales, esta tarea resulta mucho más complicada, dado que puede que participen en la ejecución varios sitios. El fallo de alguno de estos sitios, o el de un enlace de comunicaciones que conecte esos sitios, puede dar lugar a cálculos erróneos.

En este apartado se estudia la estructura del sistema de las bases de datos distribuidas y sus posibles modos de fallo. Con base en el modelo presentado en este apartado, en el Apartado 19.4 se estudian los protocolos para asegurar el compromiso atómico de las transacciones globales, y en el Apartado 19.5 se estudian los protocolos para el control de la concurrencia en las bases de datos distribuidas. En el Apartado 19.6 se estudia el modo en que pueden seguir funcionando las bases de datos distribuidas incluso en presencia de varios tipos de fallo.

19.3.1. Estructura del sistema

Cada sitio tiene su propio gestor *local* de transacciones, cuya función es asegurar las propiedades ACID de las transacciones que se ejecuten en ese sitio. Los diferentes gestores de transacciones colaboran para ejecutar las transacciones globales. Para comprender el modo en que se pueden implementar estos gestores, considérese un modelo abstracto de sistema de transacciones, en el que cada sitio contenga dos subsistemas:

- El **gestor de transacciones** administra la ejecución de las transacciones (o subtransacciones) que tienen acceso a los datos almacenados en un sitio local. Téngase en cuenta que cada una de esas transacciones puede ser una transacción local (es decir, una transacción que se ejecuta sólo en ese sitio) o parte de una transacción global (es decir, una transacción que se ejecuta en varios sitios).
- El **coordinador de transacciones** coordina la ejecución de las diferentes transacciones (tanto locales como globales) iniciadas en ese sitio.

La arquitectura global del sistema aparece en la Figura 19.1.

La estructura de los gestores de transacciones es parecida en muchos aspectos a la de los sistemas centralizados. Cada gestor de transacciones es responsable de

- Mantenimiento de un registro histórico con fines de recuperación
- Participación en un esquema adecuado de control de la concurrencia para coordinar la ejecución concurrente de las transacciones que se ejecuten en ese sitio

Como se verá, hay que modificar tanto el esquema de recuperación como el de concurrencia para adaptarlos a la distribución de las transacciones.

El subsistema del coordinador de transacciones no se necesita en los entornos centralizados, dado que las transacciones sólo tienen acceso a los datos en un sitio. Los coordinadores de transacciones, como su propio nombre implica, son responsables de la coordinación de la ejecución de todas las transacciones iniciadas en ese sitio. En cada una de esas transacciones el coordinador es responsable de

- Inicio de la ejecución de la transacción
- División de la transacción en varias subtransacciones y distribución de esas subtransacciones a los sitios correspondientes para su ejecución
- Coordinación de la terminación de la transacción, que puede hacer que la transacción se comprometa en todos los sitios o que se aborte en todos los sitios

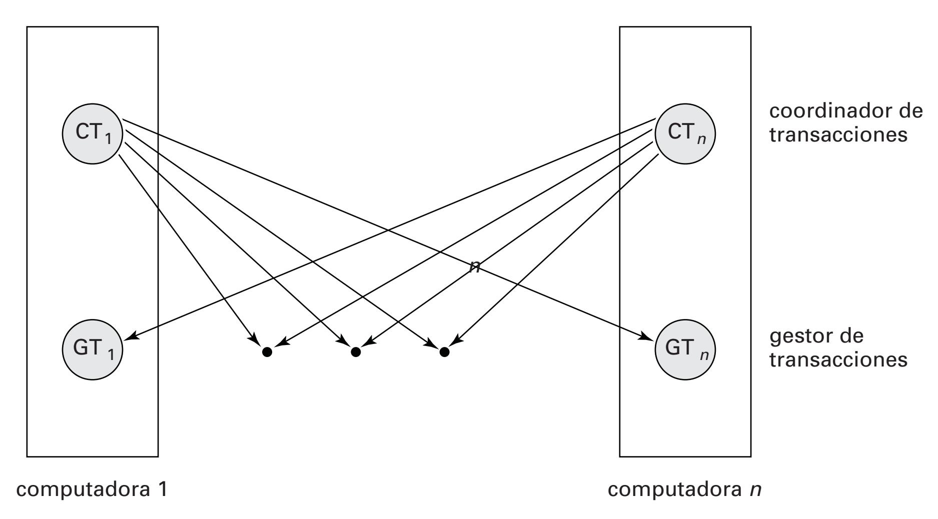


FIGURA 19.1. Arquitectura del sistema.

19.3.2. Modos de fallo del sistema

Los sistemas distribuidos pueden sufrir los mismos tipos de fallos que los sistemas centralizados (por ejemplo, errores de software, errores de hardware y fallos de discos). No obstante, hay más tipos de fallos con los que hay que tratar en los entornos distribuidos. Los tipos básicos de fallos son

- Fallo de un sitio
- Pérdida de mensajes
- Fallo de un enlace de comunicaciones
- División de la red

La pérdida o deterioro de los mensajes siempre constituye una posibilidad en los sistemas distribuidos. El sistema utiliza protocolos de control de las transmisiones, como TCP/IP, para tratar esos errores. Se puede encontrar información sobre esos protocolos en los libros de texto estándar sobre redes (véanse las notas bibliográficas).

No obstante, si dos sitios *A* y *B* no se hallan conectados de manera directa, los mensajes de uno a otro deben *encaminarse* mediante una serie de enlaces de comunicaciones. Si falla un enlace de comunicaciones los mensajes que se deberían haber transmitido por el enlace deben reencaminarse. En algunos casos resulta posible hallar otra ruta por la red de modo que los mensajes puedan alcanzar su destino. En otros casos el fallo puede hacer que no halla conexión entre algunos pares de sitios. Un sistema está **dividido** si se ha dividido en dos (o más) subsistemas, denominados **particiones**, que carecen de conexión entre ellas. Téngase en cuenta que, con esta definición, cada subsistema puede consistir en un solo nodo.

19.4. PROTOCOLOS DE COMPROMISO

Si hay que asegurar la atomicidad, todos los sitios en los que se ejecute una transacción *T* deben coincidir en el resultado final de la ejecución. *T* debe comprometerse en todos los sitios o abortarse en todos los sitios. Para asegurar esta propiedad el coordinador de transacciones de *T* debe ejecutar un *protocolo de compromiso*.

Entre los protocolos de compromiso más sencillos y más utilizados está el **protocolo de compromiso de dos fases (C2F)**, que se describe en el Apartado 19.4.1. Una alternativa es el **protocolo de compromiso de tres fases (C3F)**, que evita ciertos inconvenientes del protocolo C2F pero añade complejidad y sobrecarga. El Apartado 19.4.2 describe brevemente el protocolo C3F.

19.4.1. Compromiso de dos fases

En primer lugar se describe el modo en que opera el protocolo de compromiso de dos fases (C2F) durante el funcionamiento normal, luego describe el modo en que maneja los fallos y, finalmente, la manera en que ejecuta la recuperación y el control de la concurrencia.

Considérese una transacción *T* iniciada en el sitio S_i, en que el coordinador de transacciones es C_i.

19.4.1.1. El protocolo de compromiso

Cuando *T* completa su ejecución (es decir, cuando todos los sitios en los que se ha ejecutado *T* informan a *Ci* de que *T* se ha completado) *Ci* inicia el protocolo C2F.

- **Fase 1.** *Ci* añade el registro <*T* preparar> al registro histórico y obliga a guardar el registro histórico en un lugar de almacenamiento estable. Entonces envía un mensaje preparar *T* a todos los sitios en los que se ha ejecutado *T*. Al recibir este mensaje el gestor de transacciones del sitio determina si desea com-

prometer su parte de T. Si la respuesta es negativa, añade un registro <no T> al registro histórico y responde enviando a C_i el mensaje abortar T. Si la respuesta es sí, añade un registro <T preparada> al registro histórico y obliga a que el registro histórico (con todos los registros del registro histórico correspondientes a T) se guarde en un almacenamiento estable. El gestor de transacciones contesta entonces a Ci con el mensaje T preparada.

- **Fase 2.** Cuando Ci recibe las respuestas al mensaje preparar T de todos los sitios, o cuando ha transcurrido un intervalo de tiempo especificado con anterioridad desde que se envió el mensaje preparar T, C_i puede determinar si la transacción T puede comprometerse o abortarse. La transacción T puede comprometerse si C_i ha recibido el mensaje T preparada de todos los sitios participantes. En caso contrario, la transacción T debe abortarse. En función del resultado, se añade al registro histórico un registro <T comprometida> o un registro <T abortada> y se obliga a que el registro histórico se guarde en un almacenamiento estable. En este momento el destino de la transacción ya se ha sellado. A partir de este momento el coordinador envía un mensaje comprometer T o abortar T a todos los sitios participantes. Cuando un sitio recibe ese mensaje, lo guarda en el registro histórico.

Los sitios en los que se ejecutó T pueden abortarla de manera incondicional en cualquier momento antes de enviar el mensaje T preparada al coordinador. Una vez enviado el mensaje, se dice que la transacción está en **estado preparado** en el sitio. El mensaje T preparada constituye, en realidad, un compromiso del sitio de seguir la orden del coordinador de comprometer T o de abortarla. Para realizar ese compromiso primero hay que guardar en un almacenamiento estable la información necesaria. En caso contrario, si el sitio fallara tras enviar el mensaje T preparada, puede que no fuera capaz de cumplir su promesa. Además, los bloqueos adquiridos por la transacción deben mantenerse hasta que se complete la transacción.

Dado que se exige la unanimidad para comprometer una transacción, el destino de T queda sellado en cuanto un sitio responda abortar T. Dado que el sitio coordinador S_i es uno de los sitios en los que se ha ejecutado T, el coordinador puede decidir unilateralmente abortar T. El veredicto final sobre T se determina en el momento en que el coordinador lo escribe (comprometer o abortar) en el registro histórico y obliga a guardar el veredicto en un almacenamiento estable. En algunas implementaciones del protocolo C2F un sitio envía el mensaje acuse-de-recibo T al coordinador al final de la segunda fase del protocolo. Cuando el coordinador recibe el mensaje acuse-de-recibo T de todos los sitios, añade el registro <T completada> al registro histórico.

19.4.1.2. Tratamiento de los fallos

El protocolo C2F responde de modo diferente a diversos tipos de fallos:

- **Fallo de un sitio participante.** Si el coordinador C_i detecta que un sitio ha fallado emprende las acciones siguientes. Si el sitio falla antes de responder a C_i con el mensaje T preparada, el coordinador da por supuesto que ha respondido con el mensaje abortar T. Si el sitio falla después de que el coordinador haya recibido del sitio el mensaje T preparada, el coordinador ejecuta el resto del protocolo de compromiso de manera normal, ignorando el fallo del sitio.

  Cuando un sitio participante S_k se recupera de un fallo debe examinar su registro histórico para determinar el destino de las transacciones que se hallaban en trance de ejecución cuando se produjo el fallo. Supóngase que T es una de esas transacciones. Se toman en consideración cada uno de los casos posibles:

  — El registro histórico contiene un registro <T comprometida>. En este caso el sitio ejecuta rehacer(T).

  — El registro histórico contiene un registro <T abortada>. En este caso el sitio ejecuta deshacer(T).

  — El registro histórico contiene un registro <T preparada>. En este caso el sitio debe consultar con C_i para determinar el destino de T. Si C_i está activo, notifica a S_k si T se comprometió o se abortó. En el primer caso, ejecuta rehacer(T); en el segundo, ejecuta deshacer(T). Si C_i no está activo, S_k debe intentar hallar el destino de T consultando a otros sitios. Lo hace enviando un mensaje consulta-estado T a todos los sitios del sistema. Al recibir ese mensaje cada sitio debe consultar su registro histórico para determinar si allí se ejecutó T y, en caso afirmativo, si se comprometió o abortó. Luego notifica a S_k el resultado. Si ningún sitio tiene la información correspondiente (es decir, si T se comprometió o se abortó), S_k no puede abortar ni comprometer T. La decisión sobre T se pospone hasta que S_k pueda obtener la información necesaria. Por tanto, S_k debe volver a enviar de manera periódica el mensaje consulta-estado a los demás sitios. Lo sigue haciendo hasta que un sitio que contenga la información necesaria se recupere. Téngase en cuenta que el sitio en el que reside C_i siempre tiene la información necesaria.

  — El registro histórico no contiene ningún registro de control (abortada, comprometida, preparada) relativo a T. Por tanto, se sabe que S_k falló antes de responder al mensaje preparar T de C_i. Dado que el fallo de S_k evita el envío de la respuesta, de acuerdo con el algoritmo, C_i debe abortar T. Por tanto, S_k debe ejecutar deshacer(T).

- **Fallo del coordinador.** Si el coordinador falla durante la ejecución del protocolo de compromiso para la transacción T, los sitios participantes deben decidir el destino de T. Se verá que, en ciertos casos, los sitios participantes no pueden decidir si comprometer o abortar T, y, por tanto, deben esperar a la recuperación del coordinador que ha fallado.

  — Si un sitio activo contiene un registro <T comprometida> en su registro histórico, T debe comprometerse.

  — Si un sitio activo contiene un registro <T abortada> en su registro histórico, T debe abortarse.

  — Si algún sitio activo *no* contiene el registro <T preparada> en su registro histórico, el coordinador C_i que ha fallado no puede haber decidido comprometer T, ya que un sitio que no contenga el registro <T preparada> en su registro histórico no puede haber enviado el mensaje T preparada a C_i. No obstante, puede que el coordinador haya decidido abortar T, pero no comprometer T. En vez de esperar a que se recupere C_i, resulta preferible abortar T.

  — Si no se da ninguno de los casos anteriores todos los sitios activos deben tener un registro <T preparada> en sus registros, pero ningún otro registro de control (como <T abortada> o <T comprometida>). Dado que el coordinador ha fallado, resulta imposible determinar si se ha tomado alguna decisión, y en caso de haberse tomado, averiguar la que era, hasta que se recupere el coordinador. Por tanto, los sitios activos deben esperar a que se recupere C_i. Dado que el destino de T sigue siendo dudoso, puede seguir consumiendo recursos del sistema. Por ejemplo, si se emplean bloqueos, T puede conservar los bloqueos sobre los datos en los sitios activos. Esta situación no es deseable, dado que pueden pasar horas o días antes de que C_i vuelva a estar activo. Durante ese tiempo puede que otras transacciones se vean obligadas a esperar a T. En consecuencia, puede que los elementos de datos no estén disponibles, no sólo en el sitio que ha fallado (C_i), sino también en los sitios activos. Esta situación se denomina problema del **bloqueo**, ya que T queda bloqueada mientras espera la recuperación del sitio C_i.

- **División de la red.** Cuando una red queda dividida caben dos posibilidades:

  1. El coordinador y todos los sitios participantes siguen en una de las particiones. En este caso, el fallo no tiene ningún efecto sobre el protocolo de compromiso.
  2. El coordinador y los participantes quedan en varias particiones. Desde el punto de vista de los sitios de una de las particiones, parece como si los sitios de las demás particiones hubieran fallado. Los sitios que no se hallan en la partición que contiene al coordinador sencillamente ejecutan el protocolo para tratar el fallo del coordinador. El coordinador y los sitios que se hallan en su misma partición siguen el protocolo de compromiso habitual, dando por supuesto que los sitios de las demás particiones han fallado.

Por tanto, el mayor inconveniente del protocolo C2F es que el fallo del coordinador puede dar lugar a un bloqueo, en el que puede que haya que retrasar la decisión sobre comprometer o abortar T hasta que se recupere C_i.

19.4.1.3. Recuperación y control de la concurrencia

Cuando se reinicia un sitio que ha fallado se puede llevar a cabo la recuperación, por ejemplo, utilizando el algoritmo de recuperación descrito en el Apartado 17.9. Para tratar con los protocolos de compromiso distribuidos (como C2F y C3F),el procedimiento de recuperación debe tratar de manera especial las **transacciones dudosas**; las transacciones dudosas son transacciones para las que no se encuentra ningún registro < T preparada>, <T comprometida> ni <T abortada> en el registro histórico. El sitio que se recupera debe determinar la situación comprometer-abortar de esas transacciones, como se describe en el Apartado 19.4.1.2.

Sin embargo, si se realiza la recuperación como se acaba de describir, no puede comenzar el procesamiento normal de transacciones en el sitio hasta que se hayan comprometido o deshecho todas las transacciones dudosas. La averiguación de la situación de las transacciones dudosas puede tardar mucho tiempo, dado que puede que haya que entrar en contacto con varios sitios. Además, si ha fallado el coordinador, y ningún otro sitio tiene información sobre la situación comprometida-abortada de una transacción incompleta, se podría bloquear la recuperación si se utiliza C2F. En consecuencia, puede que el sitio que lleva a cabo la recuperación de reinicio quede inutilizable durante un largo periodo de tiempo.

Para evitar este problema los algoritmos de recuperación suelen ofrecer soporte para anotar en el registro histórico información sobre los bloqueos (se da por supuesto que se utilizan los bloqueos para el control de la concurrencia). En lugar de escribir en el registro histórico un registro <T preparada>, el algoritmo escribe en el registro histórico un registro <T preparada, L>, donde L es una lista de todos los bloqueos de escritura que tiene la transacción T cuando se escribe el registro. En el momento de la recuperación, tras llevar a cabo las acciones de recuperación, se renuevan para cada transacción dudosa T todos los bloqueos de escritura anotados en el registro <T preparada, L> del registro histórico.

Después de que se haya completado la renovación de los bloqueos para todas las transacciones dudosas

puede comenzar en el sitio el procesamiento de las transacciones, incluso antes de que se determine el estado comprometida-abortada de las transacciones dudosas. El comprometer o deshacer las transacciones dudosas se realiza de manera concurrente con la ejecución de las transacciones nuevas. Así, la recuperación del sitio es más rápida y no se bloquea nunca. Obsérvese que las transacciones nuevas que tengan un conflicto de bloqueos con cualquier bloqueo de escritura que tengan las transacciones dudosas no podrán progresar hasta que se hayan comprometido o deshecho las transacciones dudosas con las que estén en conflicto.

19.4.2. Compromiso de tres fases

El protocolo de compromiso de tres fases (C3F) es una extensión del protocolo de compromiso de dos fases que evita el problema del bloqueo con determinadas suposiciones. En concreto, se supone que no se produce ninguna fragmentación de la red y que no fallan más de *k* sitios, donde *k* es un número predeterminado. Con estas suposiciones el protocolo evita el bloqueo introduciendo una tercera fase adicional en que se implican varios sitios en la decisión de comprometer. En lugar de anotar directamente la decisión de comprometer en su almacenamiento persistente, el coordinador se asegura antes de que al menos otros *k* sitios sepan que pretende comprometer la transacción. Si el coordinador falla, los sitios restantes seleccionan primero un nuevo coordinador. Este nuevo coordinador verifica el estado del protocolo a partir de los demás sitios; si el coordinador había decidido comprometer, al menos uno de los otros *k* sitios a los que informó estará funcionando y asegurará que se respete la decisión de comprometer. El nuevo coordinador vuelve a iniciar la tercera fase del protocolo si algún sitio sabía que el antiguo coordinador pretendía comprometer la transacción. En caso contrario, el nuevo coordinador aborta la transacción.

Aunque el protocolo C3F tiene la propiedad deseable de no bloquearse a menos que fallen *k* sitios, tiene el inconveniente de que una división de la red parece lo mismo que el fallo de más de *k* sitios, lo que puede producir un bloqueo. El protocolo también tiene que implementarse con mucho cuidado para asegurarse de que la división de la red (o el fallo de más de *k* sitios) no provoque inconsistencias, de modo que una transacción se comprometa en una de las particiones y se aborte en otra. Debido a la sobrecarga que implica el protocolo C3F, no se utiliza mucho. Véanse las notas bibliográficas para hallar referencias que den más detalles del protocolo C3F.

19.4.3. Modelos alternativos del procesamiento de transacciones

Para muchas aplicaciones el problema del bloqueo del compromiso de dos fases no resulta aceptable. El problema en este caso es la idea de una sola transacción que trabaja en varios sitios. En este apartado se describe el modo de utilizar la *mensajería persistente* para evitar el problema del compromiso distribuido y luego describe brevemente el problema más importante de los *flujos de trabajo*; los flujos de trabajo se consideran con mayor detalle en el Apartado 24.2.

Para comprender la mensajería persistente considérese el modo en que se podrían transferir fondos entre dos bancos diferentes, cada uno con su propia computadora. Un enfoque es hacer que la transacción abarque los dos sitios y utilizar el compromiso de dos fases para asegurar la atomicidad. Sin embargo, puede que la transacción tenga que actualizar todo el saldo del banco y el bloqueo podría tener un grave impacto en las demás transacciones de cada banco, dado que casi todas las transacciones de los bancos actualizan el saldo total del banco.

Para comparar, considérese el modo en que se produce la transferencia de fondos mediante un cheque. El banco deduce en primer lugar el importe del cheque del saldo disponible e imprime un cheque. El cheque se transfiere físicamente al otro banco, donde se deposita. Tras comprobar el cheque, el banco aumenta el saldo local en el importe del cheque. El cheque constituye un mensaje enviado entre los dos bancos. Para que los fondos no se pierdan ni se aumenten de manera incorrecta no se debe perder el cheque ni se debe duplicar ni depositar más de una vez. Cuando las computadoras de los bancos se hallan conectadas mediante una red, los mensajes persistentes ofrecen el mismo servicio que los cheques (pero mucho más rápido, por supuesto).

Los **mensajes persistentes** son mensajes que tienen garantizada su entrega al destinatario exactamente una sola vez (ni más ni menos), independientemente de los fallos, la transacción que envía el mensaje «comprometer» tiene que ofrecer la garantía de no efectuar la entrega si la transacción se aborta. Las técnicas de recuperación de bases de datos se utilizan para implementar la mensajería persistente por encima de los canales normales de la red, como se verá en breve. A diferencia de esto, los mensajes normales pueden perderse o, incluso, entregarse varias veces en algunas circunstancias.

El manejo de los errores resulta más complicado con la mensajería persistente que con el compromiso de dos fases. Por ejemplo, si la cuenta en la que hay que ingresar el cheque se ha cerrado, hay que devolver el cheque a la cuenta que lo originó y volver a cargar su importe en ella. Por tanto, ambos sitios deben disponer de código para el manejo de errores y con código para manejar los mensajes persistentes. Sin embargo, con el compromiso de dos fases, el error lo detectaría la transacción, que no deduciría nunca el importe del cheque de la primera cuenta.

Los tipos de condiciones de excepción que pueden surgir dependen de la aplicación, por lo que no es posible que el sistema de bases de datos maneje las excepciones de manera automática. Los programas de aplicaciones que envían y reciben los mensajes persistentes

deben incluir código para manejar las condiciones de excepción y devolver el sistema a un estado consistente. Por ejemplo, no resulta aceptable que se pierda el dinero que se transfiere si se ha cerrado la cuenta receptora; hay que devolver el dinero a la cuenta ordenante, y si ello no resulta posible por algún motivo, hay que advertir a los empleados del banco para que resuelvan manualmente la situación.

Hay muchas aplicaciones en que la ventaja de la eliminación del bloqueo merece ampliamente el esfuerzo adicional de implementar los sistemas que utilizan los mensajes persistentes. De hecho, pocas organizaciones aceptarían soportar el compromiso de dos fases para transacciones que se originen fuera de ellas, dado que los fallos pueden producir el bloqueo del acceso a los datos locales. La mensajería persistente, por tanto, desempeña un papel importante en la ejecución de las transacciones que atraviesan las fronteras de las organizaciones.

Los *flujos de trabajo* proporcionan un modelo general de procesamiento de las transacciones que implican a varios sitios y, posiblemente, el procesamiento manual por los empleados de determinadas fases del proceso. Por ejemplo, cuando un banco recibe la solicitud de un crédito hay muchos pasos que debe dar, incluido el contacto con agencias externas de calificación de créditos, antes de aceptar o rechazar la solicitud. Los pasos, en su conjunto, forman un flujo de trabajo. Los flujos de trabajo se estudian con mayor detalle en el Apartado 2.4.2. También se observa que la mensajería consistente forma la base subyacente a los flujos de trabajo en los entornos distribuidos.

Se considerará ahora la **implementación** de la mensajería persistente. La mensajería persistente pueden implementarla los siguientes protocolos por encima de estructuras de mensajería que no sean dignas de confianza, que pueden perder mensajes o entregarlos varias veces:

- **Protocolo de sitio enviante.** Cuando una transacción desea enviar un mensaje persistente escribe un registro que contiene el mensaje en una relación especial *mensajes-a-enviar*, en lugar de enviar el mensaje directamente. También se le da al mensaje un identificador de mensaje único.

  Un *proceso de entrega de mensajes* controla la relación y, cuando se halla un mensaje nuevo, lo envía a su destino. Los mecanismos habituales de control de la concurrencia de las bases de datos aseguran que el proceso del sistema lea el mensaje tan sólo una vez que la transacción que escribió el mensaje se haya comprometido; si la transacción se aborta, el mecanismo habitual de recuperación borra el mensaje de la relación.

  El proceso de eliminación de mensajes sólo elimina los mensajes de la relación una vez que ha recibido un acuse de recibo del sitio de destino. Si no lo hace, pasado cierto tiempo vuelve a enviar el mensaje. Repite este proceso hasta que recibe un acuse de recibo. En caso de fallo permanente el sistema decide, pasado algún tiempo, que el mensaje no puede entregarse. Entonces se invoca al código para el manejo de excepciones proporcionado por la aplicación para tratar el fallo.

  La escritura del mensaje en una relación y su procesamiento tan sólo después de comprometer la transacción asegura que el mensaje se entregue si y sólo si la transacción se compromete. Su envío repetido garantiza que se entregue aunque haya fallos (temporales) del sistema o de la red.

- **Protocolo de sitio receptor.** Cuando un sitio recibe un mensaje persistente ejecuta una transacción que añade el mensaje a la relación especial *mensajes-recibidos*, siempre que no se halle ya presente en la relación (el identificador único de mensajes detecta los duplicados). Una vez que la transacción se compromete, o si el mensaje ya se hallaba en la relación, el sitio receptor devuelve un acuse de recibo al sitio enviante.

  Hay que tener en cuenta que enviar el acuse de recibo antes de que la transacción se comprometa no resulta seguro, ya que un fallo del sistema pudiera hacer que el mensaje se perdiera. La comprobación de si el mensaje se ha recibido previamente resulta fundamental para evitar entregas múltiples del mensaje.

  En muchos sistemas de mensajería resulta posible que los mensajes se retrasen de manera arbitraria, aunque tales retrasos sean muy improbables. Por tanto, para asegurarse, no se debe eliminar nunca el mensaje de la relación *mensajes-recibidos*. Su eliminación pudiera hacer que no se detectara una entrega duplicada. Pero, como consecuencia, la relación *mensajes-recibidos* puede crecer de manera indefinida. Para resolver este problema se da a cada mensaje una marca temporal y, si la marca temporal de un mensaje recibido es más antigua que algún punto arbitrario de corte, se descarta el mensaje. Todos los mensajes de la relación *mensajes-recibidos* que sean más antiguos que el punto de corte pueden eliminarse.

19.5. CONTROL DE LA CONCURRENCIA EN LAS BASES DE DATOS DISTRIBUIDAS

En este apartado se muestra el modo en que algunos de los esquemas de control de la concurrencia que se discuten en el Capítulo 16 pueden modificarse para que puedan utilizarse en entornos distribuidos. Se da por supuesto que cada sitio participa en la ejecución de un protocolo de compromiso para asegurar la atomicidad global de las transacciones.

Los protocolos que se describen en este apartado necesitan que se hagan actualizaciones de todas las réplicas de los elementos de datos. Si ha fallado algún sitio que contiene una réplica de un elemento de datos, no se pueden procesar las actualizaciones del elemento de datos. En el Apartado 19.6 se describen los protocolos que pueden continuar el procesamiento de las transacciones aunque haya fallado algún sitio o algún enlace, lo que proporciona una gran disponibilidad.

19.5.1. Protocolos de bloqueo

Los diferentes protocolos de bloqueo descritos en el Capítulo 16 se pueden utilizar en entornos distribuidos. La única modificación que hay que incorporar es el modo en que el gestor de bloqueos trata los datos replicados. Se presentan varios esquemas posibles que son aplicables a entornos en que los datos puedan replicarse en varios sitios. Al igual que en el Capítulo 16 se dará por supuesto la existencia de los modos de bloqueo *compartido* y *exclusivo*.

19.5.1.1. Enfoque de gestor único de bloqueos

En el enfoque de **gestor único de bloqueos** el sistema mantiene un *único* gestor de bloqueos que reside en un sitio *único* escogido (digamos S_i). Todas las solicitudes de bloqueo y de desbloqueo se realizan en el sitio S_i. Cuando una transacción necesita bloquear un elemento de datos envía una solicitud de bloqueo a S_i. El gestor de bloqueos determina si se puede conceder el bloqueo de manera inmediata. Si se puede conceder el bloqueo, el gestor de bloqueos envía un mensaje con ese objeto al sitio en el que se inició la solicitud de bloqueo. En caso contrario, la solicitud se retrasa hasta que puede concederse, en cuyo momento se envía un mensaje al sitio en el que se inició la solicitud de bloqueo. La transacción puede leer el elemento de datos de *cualquiera* de los sitios en los que residen las réplicas del elemento de datos. En caso de una operación de escritura deben implicarse en la escritura todos los sitios en los que residen réplicas del elemento de datos.

El esquema tiene las ventajas siguientes:

- **Implementación sencilla.** Este esquema necesita dos mensajes para tratar las solicitudes de bloqueo y sólo uno para las de desbloqueo.
- **Tratamiento sencillo de los interbloqueos.** Dado que todas las solicitudes de bloqueo y de desbloqueo se realizan en un solo sitio, se pueden aplicar directamente a este entorno los algoritmos de tratamiento de los interbloqueos estudiados en el Capítulo 16.

Los inconvenientes del esquema son:

- **Cuello de botella.** El sitio S_i se transforma en un cuello de botella, dado que todas las solicitudes deben procesarse allí.
- **Vulnerabilidad.** Si el sitio S_i falla se pierde el controlador de la concurrencia. Se debe detener el procesamiento o utilizar un esquema de recuperación de modo que pueda asumir la administración de los bloqueos un sitio de respaldo, como se describe en el Apartado 19.6.5.

19.5.1.2. Gestor distribuido de bloqueos

Se puede lograr un compromiso entre las ventajas y los inconvenientes ya mencionados mediante el enfoque del **gestor distribuido de bloqueos**, en el que la función de gestor de bloqueos se halla distribuida entre varios sitios.

Cada sitio mantiene un gestor de bloqueos local cuya función es gestionar las solicitudes de bloqueo y de desbloqueo para los elementos de datos que se almacenan en ese sitio. Cuando una transacción desea bloquear el elemento de datos Q, que no está replicado y reside en el sitio S_i, se envía un mensaje al gestor de bloqueos del sitio S_i para solicitarle un bloqueo (en un modo de bloqueo determinado). Si el elemento de datos Q está bloqueado en un modo incompatible, se retrasa la solicitud hasta que puede concederse. Una vez se ha determinado que la solicitud de bloqueo puede concederse el gestor de bloqueos devuelve un mensaje al sitio que ha iniciado la solicitud que indica que ha concedido la solicitud de bloqueo.

Hay varios modos alternativos de tratar con la réplica de los elementos de datos, que se estudian en los apartados 19.5.1.3 a 19.5.1.6.

El esquema del gestor distribuido de bloqueos tiene la ventaja de su implementación sencilla y reduce el grado en el que el coordinador constituye un cuello de botella. Tiene una sobrecarga razonablemente baja, ya que sólo necesita dos transferencias de mensajes para tratar las solicitudes de bloqueo y una transferencia de mensaje para las de desbloqueo. No obstante, el tratamiento de los interbloqueos resulta más complejo, dado que las solicitudes de bloqueo y de desbloqueo ya no se realizan en un solo sitio. Puede haber interbloqueos entre sitios aunque no haya interbloqueos en ninguno de los

sitios. Los algoritmos de tratamiento de los interbloqueos estudiados en el Capítulo 16 deben modificarse, como se discutirá en el Apartado 19.5.4. para detectar los interbloqueos globales.

19.5.1.3. Copia principal

Cuando un sistema utiliza la réplica de datos se puede escoger una de las réplicas como **copia principal**. Así, para cada elemento de datos Q la copia principal de Q debe residir exactamente en un sitio, que se denomina **sitio principal** de Q.

Cuando una transacción necesita bloquear un elemento de datos Q solicita un bloqueo en el sitio principal de Q. Como ya se ha visto, la respuesta a la solicitud se retrasa hasta que pueda concederse.

Por tanto, la copia principal permite que el control de concurrencia de los datos replicados se trate como el de los datos no replicados. Esta semejanza permite una implementación sencilla. No obstante, si falla el sitio principal de Q, Q queda inaccesible, aunque otros sitios que contengan réplicas estén accesibles.

19.5.1.4. Protocolo de mayoría

El **protocolo de mayoría** funciona de la manera siguiente. Si el elemento de datos Q se replica en n sitios diferentes se debe enviar un mensaje de solicitud de bloqueo a más del 50 por 100 de los n sitios en los que se almacena Q. Cada gestor de bloqueos determina si se puede conceder el bloqueo de manera inmediata (en lo que a él se refiere). Como ya se ha visto, la respuesta se retrasa hasta que se pueda conceder la solicitud. La transacción no se opera en Q hasta que logre obtener un bloqueo en la mayoría de las réplicas de Q.

Por el momento se asume que las escrituras se realizan en todas las réplicas, exigiendo que todos los sitios que contengan réplicas estén disponibles. Sin embargo, la mayor ventaja del protocolo de mayoría es que se puede extender para tratar los fallos de los sitios, como se verá en el Apartado 19.6.1. Este esquema trata los datos replicados de manera descentralizada, con lo que evita los inconvenientes del control centralizado. No obstante, presenta los siguientes inconvenientes:

- **Implementación.** El protocolo de mayoría es más complicado de implementar que los esquemas anteriores. Necesita $2(n/2+1)$ mensajes para manejar las solicitudes de bloqueo y $(n/2+1)$ mensajes para manejar las solicitudes de desbloqueo.
- **Tratamiento de los interbloqueos.** Además del problema de los interbloqueos locales debidos al empleo del enfoque del gestor distribuido de bloqueos, puede que se produzca un interbloqueo aunque sólo se esté bloqueando un elemento de datos. A modo de ejemplo, considérese un sistema con cuatro sitios y réplica completa. Supóngase que las transacciones T_1 y T_2 desean bloquear el elemento de datos Q en modo exclusivo. Puede que la transacción T_1 tenga éxito en bloquear Q en los sitios S_1 y S_3, mientras que puede que la transacción T_2 consiga bloquear Q en los sitios S_2 y S_4. Cada una de ellas deberá esperar a adquirir el tercer bloqueo; por tanto, se ha producido un interbloqueo. Por fortuna, se pueden evitar estos interbloqueos con relativa facilidad exigiendo que todos los sitios soliciten los bloqueos de las réplicas de un elemento de datos en el mismo orden predeterminado.

19.5.1.5. Protocolo sesgado

El **protocolo sesgado** es otro enfoque del manejo de las réplicas. La diferencia con el protocolo de mayoría es que se concede un tratamiento más favorable a las solicitudes de bloqueos compartidos que a las solicitudes de bloqueos exclusivos.

- **Bloqueos compartidos.** Cuando una transacción necesita bloquear un elemento de datos Q simplemente solicita un bloqueo de Q al gestor de bloqueos de un sitio que contenga una réplica de Q.
- **Bloqueos exclusivos.** Cuando una transacción necesita bloquear el elemento de datos Q solicita un bloqueo de Q al gestor de bloqueos de todos los sitios que contienen una réplica de Q.

Como ya se ha visto, la respuesta a la solicitud se retrasa hasta que pueda concederse. El esquema sesgado tiene la ventaja de imponer menos sobrecarga a las operaciones de lectura que el protocolo de mayoría. Este ahorro resulta especialmente significativo en los casos frecuentes en los que la frecuencia de las operaciones de lectura es mucho mayor que la de las operaciones de escritura. No obstante, la sobrecarga adicional sobre las operaciones de escritura supone un inconveniente. Además, el protocolo sesgado comparte el inconveniente de la complejidad en el manejo de interbloqueos con el protocolo de mayoría.

19.5.1.6. Protocolo de consenso de quórum

El protocolo de **consenso de quórum** es una generalización del protocolo de mayoría. El protocolo de consenso de quórum asigna a cada sitio un peso no negativo. Asigna a las operaciones de lectura y de escritura del elemento x dos enteros, denominados **quórum de lectura** Q_l y **quórum de escritura** Q_e, que deben cumplir la condición siguiente, donde S es el peso total de todos los sitios en los que reside x:

$$Q_l + Q_e > S \textbf{ y } 2 * Q_e > S$$

Para ejecutar una operación de lectura deben bloquearse suficientes réplicas como para que su peso total sea $\geq Q_l$. Para ejecutar una operación de escritura se deben bloquear suficientes réplicas como para que su peso total sea $\geq Q_e$.

Una ventaja del enfoque de consenso de quórum es que puede permitir reducir de manera selectiva el coste de las operaciones de bloqueo de lectura o de escritura definiendo de manera adecuada los quórum de lectura y de escritura. Por ejemplo, con un quórum de lectura pequeño las operaciones de lectura necesitan obtener menos bloqueos, pero el quórum de escritura será mayor, por lo que las operaciones de escritura necesitan obtener más bloqueos. Además, si se asignan pesos más elevados a algunos sitios (por ejemplo, a los que tengan menos posibilidad de fallar), hay que tener acceso a menos sitios para adquirir los bloqueos.

De hecho, al definir los pesos y los quórum de manera adecuada, el protocolo de consenso de quórum puede simular el protocolo de mayoría y los protocolos sesgados.

19.5.2. Marcas temporales

La idea principal tras el esquema de marcas temporales del Apartado 16.2 es que se concede a cada transacción una marca temporal *única* que el sistema utiliza para decidir el orden de secuenciación. La primera tarea, por tanto, al generalizar el esquema centralizado a un esquema distribuido es desarrollar un esquema para generar marcas temporales únicas. Luego, los diferentes protocolos pueden operar directamente en el entorno no replicado.

Hay dos métodos principales para la generación de marcas temporales únicas, uno centralizado y otro distribuido. En el esquema centralizado un solo sitio distribuye las marcas temporales. El sitio puede utilizar un contador lógico o su propio reloj local con esta finalidad.

En el esquema distribuido cada sitio genera una marca temporal local única mediante un contador lógico o el reloj local. La marca temporal global única se obtiene concatenando la marca temporal local única con el identificador del sitio, que también debe ser único (Figura 19.2). El orden de concatenación es importante. El identificador del sitio se utiliza en la posición menos significativa para asegurar que las marcas temporales globales generadas en un sitio no sean siempre mayores que las generadas en otro. Compárese esta técnica para la generación de marcas temporales únicas con la presentada en el Apartado 19.2.3 para la generación de nombres únicos.

Puede que todavía surja algún problema si un sitio genera marcas temporales locales a una velocidad mayor que la de los demás sitios. En ese caso el contador lógico del sitio rápido será mayor que el de los demás sitios. Por tanto, todas las marcas temporales generadas por el sitio rápido serán mayores que las generadas por los demás sitios. Lo que se necesita es un mecanismo que asegure que las marcas temporales locales se generen de manera homogénea en todo el sistema. En cada sitio S_i se define un **reloj lógico** (RL_i), que genera la marca temporal local única. El reloj lógico puede implementarse como un contador que se incremente después de generar una nueva marca temporal local. Para asegurar que los diferentes relojes lógicos estén sincronizados se exige que el sitio S_i adelante su reloj lógico siempre que una transacción T_i con la marca temporal $<x,y>$ visite ese sitio y x sea mayor que el valor actual de RL_i. En ese caso el sito S_i adelanta su reloj lógico al valor $x + 1$.

Si se utiliza el reloj del sistema para generar marcas temporales, éstas se asignarán de manera homogénea, dado que ningún sitio tiene un reloj del sistema que vaya rápido o lento. Dado que es posible que los relojes no sean totalmente exactos, se debe utilizar una técnica similar a la de los relojes lógicos para asegurar que ningún reloj se adelante o se atrase mucho respecto de los demás.

19.5.3. Réplica con grado de consistencia bajo

Muchas bases de datos comerciales de hoy en día soportan las réplicas, que pueden adoptar varias formas. Con la **réplica maestro-esclavo** la base de datos permite las actualizaciones en el sitio principal y las propaga de manera automática a las réplicas de los demás sitios. Las transacciones pueden leer las réplicas en los demás sitios, pero no se les permite actualizarlas.

Una característica importante de esta réplica es que las transacciones no consiguen bloqueos en los sitios remotos. Para asegurar que las transacciones que se ejecutan en los sitios réplicas vean una vista consistente (aunque quizás desactualizada) de la base de datos la réplica debe reflejar una **instantánea consistente para las réplicas** de los datos del sitio principal, es decir, la réplica debe reflejar todas las actualizaciones hasta una transacción dada en el orden de secuenciación.

La base de datos puede estar configurada para propagar las actualizaciones de manera inmediata una vez

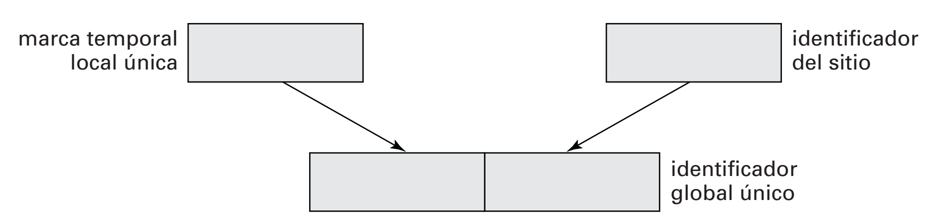


FIGURA 19.2. Generación de marcas temporales únicas.

producidas en el sitio principal o para propagarlas sólo de manera periódica.

La réplica maestro-esclavo resulta especialmente útil para distribuir información, por ejemplo desde una oficina central a las sucursales de una organización. Otro empleo de esta forma de réplica es la creación de copias de la base de datos para ejecutar consultas de gran tamaño, de modo que las consultas no interfieran con las transacciones. Las actualizaciones deben propagarse de manera periódica (cada noche, por ejemplo) de modo que la propagación no interfiera con el procesamiento de las consultas.

El sistema de bases de datos Oracle posee la sentencia **create snapshot** (crear instantánea), que puede crear una instantánea consistente para las transacciones de una relación, o de un conjunto de relaciones, en un sitio remoto. También soporta la actualización de instantáneas, que puede hacerse volviendo a calcular la instantánea o actualizándola de manera incremental. Oracle soporta la actualización automática, bien continuamente, bien a intervalos periódicos.

Con la **réplica multimaestro** (también denominada **réplica de actualización distribuida**) se permiten las actualizaciones en cualquier réplica de los elementos de datos y se propagan de manera automática a todas las réplicas. Este modelo es el modelo básico utilizado para administrar las réplicas en las bases de datos distribuidas. Las transacciones actualizan la copia local y el sistema actualiza las demás réplicas de manera transparente.

Un modo de actualizar las réplicas es aplicar la actualización inmediata con el compromiso de dos fases, utilizando una de las técnicas de control de la concurrencia distribuida que se han visto. Muchos sistemas de bases de datos utilizan el protocolo sesgado, en el que las operaciones de escritura tienen que bloquear y actualizar todas las réplicas y las operaciones de lectura bloquean y leen cualquier réplica, como técnica de control de la concurrencia.

Muchos sistemas de bases de datos ofrecen una forma alternativa de actualización: Actualizan en un sitio, con **propagación perezosa** de las actualizaciones a los demás sitios, en lugar de aplicar de manera inmediata las actualizaciones a todas las réplicas como parte de la transacción que lleva a cabo la actualización. Los esquemas basados en la propagación perezosa permiten que continúe el procesamiento de las transacciones (incluidas las actualizaciones) aunque un sitio quede desconectado de la red, lo que mejora la disponibilidad, pero, por desgracia, lo hacen a costa de la consistencia. Se suele seguir uno de estos dos enfoques cuando se utiliza la propagación perezosa:

- Las actualizaciones de las réplicas se traducen en actualizaciones del sitio principal, que se propagan luego de manera perezosa a todas las réplicas.

  Este enfoque asegura que las actualizaciones de un elemento se ordenen de manera secuencial, aunque puedan producirse problemas de secuenciabilidad, dado que las transacciones pueden leer un valor antiguo de algún otro elemento de datos y utilizarlo para llevar a cabo una actualización.
- Las actualizaciones se llevan a cabo en cualquier réplica y se propagan a todas las demás.

  Este enfoque puede causar todavía más problemas, dado que el mismo elemento de datos puede ser actualizado de manera concurrente en varios sitios.

Algunos conflictos debidos a la falta de control de la concurrencia distribuida pueden detectarse cuando se propagan a otros sitios las actualizaciones (se verá el modo en el Apartado 23.5.4), pero la resolución del conflicto implica retroceder a las transacciones que estén comprometidas y, por tanto, no se garantiza la durabilidad de las mismas. Además, puede que se necesite intervención del personal encargado para resolver los conflictos. Por tanto, los esquemas mencionados deben evitarse o utilizarse con precaución.

19.5.4. Tratamiento de los interbloqueos

Los algoritmos de prevención y de detección de interbloqueos del Capítulo 16 pueden utilizarse en los sistemas distribuidos, siempre que se realicen modificaciones. Por ejemplo, se puede utilizar el protocolo de árbol definiendo un árbol *global* entre los elementos de datos del sistema. De manera parecida, el enfoque de ordenación por marcas temporales puede aplicarse de manera directa en entornos distribuidos, como se vio en el Apartado 19.5.2.

La prevención de interbloqueos puede dar lugar a esperas y retrocesos innecesarios. Además, puede que algunas técnicas de prevención de interbloqueos necesiten que se impliquen en la ejecución de una transacción más sitios que los que serían necesarios de otro modo.

Si se permite que los interbloqueos se produzcan y se confía en su detección, el problema principal en los sistemas distribuidos es la decisión del modo en que se mantiene el grafo de espera. Las técnicas habituales para tratar este problema exigen que cada sitio guarde un **grafo local de espera**. Los nodos del grafo corresponden a todas las transacciones (locales y no locales) que en cada momento tienen o solicitan alguno de los elementos locales de ese sitio. Por ejemplo, la Figura 19.3 muestra un sistema que consta de dos sitios, cada uno de los cuales mantiene su grafo de espera. Obsérvese que las transacciones T_2 y T_3 aparecen en los dos grafos, lo que indica que han solicitado elementos en los dos sitios.

Estos grafos locales de espera se crean de la manera habitual para las transacciones y los elementos de datos locales. Cuando una transacción T_i en el sitio S_1 necesita un recurso del sitio S_2, envía un mensaje de solicitud al sitio S_2. Si el recurso lo tiene la transacción

T_j, el sistema introduce un arco $T_i \rightarrow T_j$ en el grafo de espera local del sitio S_2.

Evidentemente, si algún grafo de espera local tiene un ciclo, se ha producido un interbloqueo. Por otro lado, el hecho de que no haya ciclos en ninguno de los grafos locales de espera no significa que no haya interbloqueos. Para ilustrar este problema considérense los grafos locales de espera de la Figura 19.3. Cada grafo de espera es acíclico y, sin embargo, hay un interbloqueo en el sistema debido a que la *unión* de los grafos locales de espera contiene un ciclo. Este grafo aparece en la Figura 19.4.

En el enfoque de **detección centralizada de interbloqueos** el sistema crea y mantiene un **grafo global de espera** (la unión de todos los grafos locales) en un *solo* sitio: el coordinador de detección de interbloqueos. Dado que hay un retraso en las comunicaciones en el sistema hay que distinguir entre dos tipos de grafos de espera. Los grafos *reales* describen el estado real pero desconocido del sistema en cualquier momento dado, como lo vería un observador omnisciente. Los grafos *creados* son una aproximación generada por el controlador durante la ejecución del algoritmo del controlador. Evidentemente, el controlador debe generar el grafo creado de modo que, siempre que se invoque al algoritmo de detección, los resultados mostrados sean correctos. *Correcto* significa en este caso que, si hay algún interbloqueo, se comunique con prontitud y, si el sistema comunica algún interbloqueo, realmente se halle en estado de interbloqueo.

El grafo global de espera debe poder volver a crearse o actualizarse bajo las siguientes condiciones:

- Siempre que se introduzca o se elimine un nuevo arco en alguno de los grafos locales de espera.
- De manera periódica, cuando se hayan producido varias modificaciones en un grafo local de espera.
- Siempre que el coordinador necesite invocar el algoritmo de detección de ciclos.

Cuando el coordinador invoca el algoritmo de detección de interbloqueo busca en su grafo global. Si halla un ciclo, selecciona una víctima para hacerla retroceder. El coordinador debe comunicar a todos los sitios que se ha seleccionado como víctima a una transacción concreta. Los sitios, a su vez, hacen retroceder la transacción víctima.

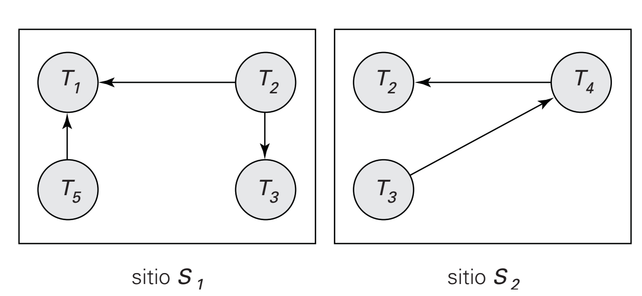


FIGURA 19.3. Grafos locales de espera.

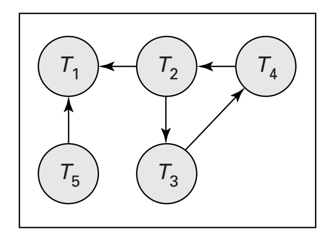


FIGURA 19.4. Grafo global de espera de la Figura 19.3.

Este esquema puede producir retrocesos innecesarios si:

- Hay **ciclos falsos** en el grafo global de espera. A modo de ejemplo, considérese una instantánea del sistema representado por los grafos locales de espera de la Figura 19.5. Supóngase que T_2 libera el recurso que tiene en el sitio S_1, lo que provoca la eliminación del arco $T_1 \rightarrow T_2$ en S_1. La transacción T_2 solicita entonces un recurso que tiene T_3 en el sitio S_2, lo que provoca la agregación del arco $T_2 \rightarrow T_3$ en S_2. Si el mensaje insertar $T_2 \rightarrow T_3$ de S_2 llega antes que el mensaje eliminar $T_1 \rightarrow T_2$ de S_1, puede que el coordinador descubra el falso ciclo $T_1 \rightarrow T_2 \rightarrow T_3$ después del mensaje insertar (pero antes del mensaje eliminar). Puede que se inicie la recuperación de interbloqueo, aunque no se haya producido ninguno.

  Obsérvese que la situación con ciclos falsos no puede producirse con bloqueos de dos fases. La probabilidad de los ciclos falsos suele ser lo bastante baja como para que no cause un problema serio de rendimiento.

- Se produce de verdad un *interbloqueo* y se escoge una víctima cuando se aborta alguna de las tran-

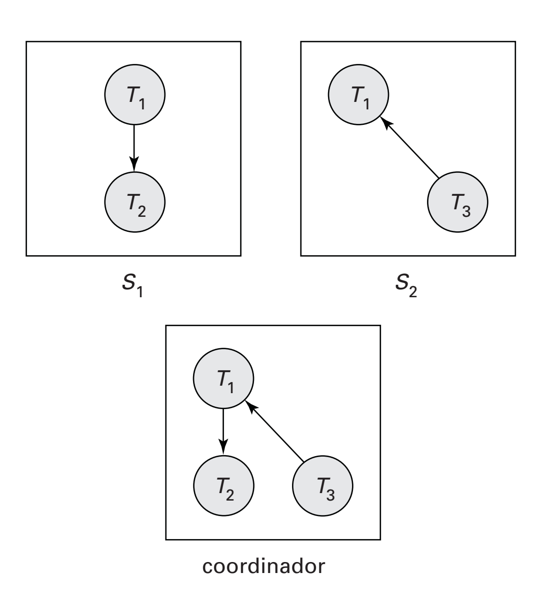


FIGURA 19.5. Ciclos falsos en el grafo global de espera.

sacciones por motivos no relacionados con interbloqueos. Por ejemplo, supóngase que el sitio S_1 de la Figura 19.3 decide abortar T_2. Al mismo tiempo, el coordinador ha descubierto un ciclo y ha escogido como víctima a T_3. Tanto T_2 como T_3 se hacen retroceder, aunque sólo hiciera falta hacer retroceder a T_2.

La detección de interbloqueos puede hacerse de manera distribuida, con varios sitios que asuman partes de la tarea, en lugar de hacerla en un solo sitio. No obstante, estos algoritmos resultan más complicados y más costosos. Véanse las notas bibliográficas para hallar referencias a estos algoritmos

19.6. DISPONIBILIDAD

Uno de los objetivos del empleo de bases de datos distribuidas es la **disponibilidad elevada**; es decir, la base de datos debe funcionar casi todo el tiempo. En concreto, dado que los fallos son más probables en los sistemas distribuidos de gran tamaño, una base de datos distribuida debe seguir funcionando aunque haya varios tipos de fallos. La capacidad de continuar funcionando incluso durante los fallos se denomina **robustez**.

Para que un sistema distribuido sea robusto debe *detectar* los fallos, *reconfigurar* el sistema de modo que el cálculo pueda continuar y *recuperarse* cuando se repare el procesador o el enlace.

Los diferentes tipos de fallos se tratan de manera diferente. Por ejemplo, la pérdida de mensajes se trata mediante su retransmisión. La retransmisión repetida de un mensaje por un enlace sin la recepción de un acuse de recibo suele ser síntoma de fallo del enlace. La red suele intentar hallar una ruta alternativa para el mensaje. La incapacidad de hallar esa ruta suele ser síntoma de división de la red.

No obstante, no suele ser posible diferenciar claramente entre los fallos de los sitios y la división de la red. El sistema suele poder detectar que se ha producido un fallo, pero puede que no logre identificar el tipo de fallo. Por ejemplo, supóngase que el sitio S_1 no puede comunicar con S_2. Puede ser que S_2 haya fallado. No obstante, otra posibilidad es que el enlace entre S_1 y S_2 haya fallado, lo que provoca la división de la red. El problema se aborda en parte empleando varios enlaces entre los sitios, de modo que aunque falle un enlace los sitios sigan conectados. Sin embargo, todavía puede producirse un fallo de varios enlaces, por lo que hay situaciones en las que no se puede estar seguro de si se ha producido un fallo del sitio o una división de la red.

Supóngase que el sitio S_1 ha descubierto que se ha producido un fallo. Debe iniciar un procedimiento que permita que el sistema se reconfigure y continúe con el modo normal de operación.

- Si en el momento del fallo había transacciones activas en un sitio que haya fallado o que haya quedado inaccesible, esas transacciones se deben abortar. Resulta conveniente abortar con prontitud esas transacciones, ya que tienen bloqueos sobre los datos en sitios que siguen activos; esperar a que el sitio que ha fallado o que ha quedado inaccesible vuelva a estar accesible puede impedir otras transacciones en sitios que están operativos.

  No obstante, en algunos casos, cuando los objetos de datos están replicados, puede que sea posible seguir con las operaciones de lectura y de actualización aunque algunas réplicas estén inaccesibles. En ese caso, cuando se recupera el sitio que ha fallado, si tenía réplicas de algún objeto de datos, debe obtener los valores actualizados de esos objetos de datos y asegurarse de que recibe todas las actualizaciones posteriores. Este problema se aborda en el Apartado 19.6.1.
- Si los datos replicados se guardan en un sitio que ha fallado o que está inaccesible, el catálogo debe actualizarse para que las consultas no hagan referencia a la copia ubicada en ese sitio. Cuando un sitio vuelve a estar activo hay que asegurarse de que los datos que haya en él sean consistentes, como se verá en el Apartado 19.6.3.
- Si un sitio que ha fallado es el servidor central de algún subsistema hay que celebrar una *elección* para determinar el nuevo servidor (véase el Apartado 19.6.5). Entre los servidores centrales están los servidores de nombres, los coordinadores de concurrencia y los detectores globales de interbloqueo.

Dado que, en general, no es posible distinguir entre los fallos de los enlaces de red y los fallos de los sitios, cualquier esquema de reconfiguración debe estar diseñado para funcionar de manera correcta en caso de división de la red. En concreto, deben evitarse las situaciones siguientes:

- Que se elijan dos o más servidores centrales en particiones distintas.
- Que más de una partición actualice un elemento de datos replicado.

19.6.1. Enfoque basado en la mayoría

El enfoque basado en la mayoría del control distribuido de la concurrencia del Apartado 19.5.1.4 puede modificarse para que funcione a pesar de los fallos. En este enfoque, cada objeto de datos guarda con él un número de versión para detectar cuándo se produjo la última

operación de escritura en él. Siempre que una transacción escribe un objeto también actualiza el número de versión de la manera siguiente:

- Si el objeto de datos *a* se replica en *n* sitios diferentes, se debe enviar un mensaje de solicitud de bloqueo a más de la mitad de los *n* sitios en los que se guarda *a*. La transacción no opera sobre *a* hasta que ha conseguido obtener un bloqueo en la mayoría de las réplicas de *a*.
- Las operaciones de lectura examinan todas las réplicas sobre las que se ha obtenido el bloqueo y leen el valor de la réplica que tiene el número de versión más elevado (de manera opcional también pueden escribir este valor en las réplicas con números de versión más bajos). Las operaciones de escritura leen todas las réplicas igual que hacen las operaciones de lectura para hallar el número de versión más elevado (este paso de la transacción normalmente lo habrá realizado antes una operación de lectura, y se puede volver a utilizar el resultado). El número de versión nuevo es una unidad mayor que el número de versión más elevado. La operación de escritura escribe en todas las réplicas en las que ha obtenido bloqueos y define el nuevo número de versión como número de versión de todas las réplicas.

Los fallos durante las transacciones (tanto las divisiones de la red como los fallos de los sitios) pueden tolerarse siempre que 1) los sitios disponibles en compromiso contengan la mayoría de las réplicas de todos los objetos en los que hay que escribir y 2) durante las operaciones de lectura se lea la mayoría de las réplicas para hallar los números de versión. Si se violan estos requisitos se debe abortar la transacción. Siempre que se satisfagan los requisitos se puede utilizar el protocolo de compromiso de dos fases, como siempre, en los sitios que estén disponibles.

En este esquema la reintegración resulta trivial; no hay que hacer nada. Esto se debe a que las operaciones de escritura han actualizado la mayoría de las réplicas, mientras las operaciones de lectura leen la mayoría de las réplicas y hallan como mínimo una que tenga la última versión.

La técnica de numeración de versiones utilizada con el protocolo de mayoría también puede utilizarse para hacer que funcione el protocolo de consenso de quórum en presencia de fallos. Los detalles (evidentes) se dejan al lector. No obstante, el riesgo de que los fallos eviten que el sistema procese las transacciones aumenta si se asignan pesos superiores a algunos sitios.

19.6.2. Enfoque leer uno, escribir todos los disponibles

Como caso especial de consenso de quórum se puede emplear el protocolo sesgado asignando pesos unitarios a todos los sitios, definiendo el quórum de lectura como 1 y definiendo el quórum de escritura como *n* (todos los sitios). En este caso especial no hace falta utilizar números de versión; sin embargo, con que falle un solo sitio que contenga un elemento de datos no podrá llevarse a cabo ninguna operación de escritura en el elemento, dado que no se dispondrá del quórum de escritura. Este protocolo se denomina protocolo **leer uno, escribir todos**, ya que hay que escribir todas las réplicas.

Para permitir que continúe el trabajo en caso de fallos sería conveniente poder utilizar un protocolo **lectura de uno, escritura de todos los disponibles**. En este enfoque las operaciones de lectura se llevan a cabo como en el esquema **leer uno, escribir todos**; se puede leer cualquier réplica disponible y se obtienen un bloqueo de lectura sobre esa réplica. Se envía una operación de escritura a todas las réplicas y se adquieren bloqueos de escritura sobre todas las réplicas. Si un sitio no está disponible, el gestor de transacciones continúa su labor sin esperar a que el sitio se recupere.

Aunque este enfoque parezca muy atractivo, presenta varias complicaciones. En concreto, los fallos de comunicación temporales pueden hacer que un sitio parezca no disponible, haciendo que no se lleve a cabo la operación de escritura pero, cuando el enlace se restaura, el sitio no sabe que tiene que llevar a cabo acciones de reintegración para ponerse al día con las operaciones de escritura que ha perdido. Además, si la red se fragmenta, puede que cada partición pase a actualizar el mismo elemento de datos, creyendo que los sitios de las demás particiones no funcionan.

El esquema leer uno, escribir todos los disponibles puede utilizarse si nunca se producen divisiones de la red, pero puede dar lugar a inconsistencias en caso de que se produzcan esas fragmentaciones.

19.6.3. Reintegración de los sitios

La reintegración al sistema de los sitos o de los enlaces reparados exige la adopción de precauciones. Cuando se recupera un sitio que ha fallado, debe iniciar un procedimiento para actualizar sus tablas del sistema para que reflejen las modificaciones realizadas mientras estaba fuera de servicio. Si el sitio tiene réplicas de elementos de datos, debe obtener los valores actualizados de esos elementos de datos y asegurarse de que recibe todas las actualizaciones que se produzcan a partir de entonces. La reintegración de los sitios es más complicada de lo que parece a primera vista, dado que puede que haya actualizaciones de los elementos de datos que se hayan procesado durante el tiempo en el que el sitio se está recuperando.

Una solución sencilla es detener temporalmente todo el sistema hasta que el sitio que ha fallado se vuelva a unir a él. En la mayor parte de las aplicaciones, sin embargo, esa detención temporal plantea problemas inaceptables. Se han desarrollado técnicas para permitir

que los sitios que han fallado se reintegren mientras se ejecutan de manera concurrente las actualizaciones de los elementos de datos. Antes de que se conceda ningún bloqueo de lectura o de escritura sobre cualquier elemento de datos, el sitio debe asegurarse de que se ha puesto al día con todas las actualizaciones del elemento de datos. Si se recupera un enlace que había fallado se pueden volver a unir dos o más particiones. Dado que la división de la red limita las operaciones admisibles para algunos de los sitios o para todos ellos, se debe informar con prontitud a todos los sitios de la recuperación del enlace. Véanse las notas bibliográficas para obtener más información sobre la recuperación en los sistemas distribuidos.

19.6.4. Comparación con la copia de seguridad remota

Los sistemas remotos de copia de seguridad, que se estudiaron en el Apartado 17.10, y la réplica en las bases de datos distribuidas son dos enfoques alternativos para la provisión de una disponibilidad elevada. La diferencia principal entre los dos esquemas es que con los sistemas remotos de copia de seguridad las acciones como el control de la concurrencia y la recuperación se llevan a cabo en un único sitio, y sólo se replican en el otro sitio los datos y los registros del registro histórico. En concreto, los sistemas remotos de copia de seguridad ayudan a evitar el compromiso de dos fases, y las sobrecargas resultantes. Además, las transacciones sólo tienen que entrar en contacto con un sitio (el sitio principal) y, así, se evita la sobrecarga de la ejecución del código de las transacciones en varios sitios. Por tanto, los sistemas remotos de copia de seguridad ofrecen un enfoque de la elevada disponibilidad de menor coste que la réplica.

Por otro lado, la réplica puede ofrecer mayor disponibilidad teniendo disponibles varias réplicas y empleando el protocolo de mayoría.

19.6.5. Selección del coordinador

Varios de los algoritmos que se han presentado exigen el empleo de un coordinador. Si el coordinador falla debido a un fallo del sitio en el que reside el sistema, sólo puede continuar la ejecución reiniciando un nuevo coordinador en otro sitio. Un modo de continuar la ejecución es el mantenimiento de un coordinador suplente, que esté preparado para asumir la responsabilidad si el coordinador falla.

El **coordinador suplente** es un sitio que, además de otras tareas, mantiene de manera local suficiente información como para permitirle asumir el papel de coordinador con un perjuicio mínimo al sistema distribuido. Todos los mensajes dirigidos al coordinador los reciben tanto el coordinador como su suplente. El coordinador suplente ejecuta los mismos algoritmos y mantiene la misma información interna de estado (como, por ejemplo, para el coordinador de concurrencia, la tabla de bloqueos) que el coordinador auténtico. La única diferencia en funcionamiento entre el coordinador y su suplente es que el suplente no emprende ninguna acción que afecte a otros sitios. Esas acciones se dejan al coordinador auténtico.

En caso de que el coordinador suplente detecte el fallo del coordinador auténtico asume el papel de coordinador. Dado que el suplente tiene toda la información disponible que tenía el coordinador que ha fallado, el procesamiento puede continuar sin interrupción.

La ventaja principal del enfoque del suplente es la capacidad de continuar el procesamiento de manera inmediata. Si no hubiera un suplente dispuesto a asumir la responsabilidad del coordinador, el coordinador que se designara ex novo tendría que buscar la información en todos los sitios del sistema para poder ejecutar las tareas de coordinación. Con frecuencia la única fuente de parte de la información necesaria es el coordinador que ha fallado. En ese caso, puede que sea necesario abortar parte de las transacciones activas (o todas ellas) y reiniciarlas bajo el control del nuevo coordinador.

Por tanto, el enfoque del coordinador suplente evita el retraso sustancial de la espera mientras el sistema distribuido se recupera de un fallo del coordinador. El inconveniente es la sobrecarga de la ejecución duplicada de las tareas del coordinador. Además, el coordinador y su suplente necesitan comunicarse de manera regular para asegurarse de que sus actividades están sincronizadas.

En resumen, el enfoque del coordinador suplente supone una sobrecarga durante el procesamiento normal para permitir una recuperación rápida de los fallos del coordinador.

En ausencia de un coordinador suplente designado, o con objeto de tratar varios fallos, los sitios que siguen funcionando pueden escoger de manera dinámica un nuevo coordinador. Los **algoritmos de elección** permiten que los sitios escojan el sitio del nuevo coordinador de manera descentralizada. Los algoritmos de selección necesitan que se asocie un único número de identificación con cada sitio activo del sistema.

El **algoritmo luchador** para la elección funciona de la manera siguiente. Para que la notación y la discusión no dejen de ser sencillas, supóngase que el número de identificación del sitio S_i es i y que el coordinador elegido siempre será el sitio activo con el número de identificación más elevado. Por tanto, cuando un coordinador falla, el algoritmo debe elegir el sitio activo que tenga el número de identificación más elevado. El algoritmo debe enviar este número a cada sitio activo del sistema. Además, el algoritmo debe proporcionar un mecanismo por el que los sitios que se recuperen de un fallo puedan identificar al coordinador activo. Supóngase que el sitio S_i envía una solicitud que el coordinador no responde dentro de un intervalo de tiempo predeterminado T. En esa situación se supone que el

coordinador ha fallado y S_i intenta elegirse a sí mismo como sitio del nuevo coordinador.

El sitio S_i envía un mensaje de elección a cada sitio que tenga un número de identificación más elevado. Luego, el sitio S_i espera, un intervalo de tiempo T, la respuesta de cualquiera de esos sitios. Si no recibe respuesta dentro del tiempo T, da por supuesto que todos los sitios con números mayores que i han fallado, se elige a sí mismo sitio del nuevo coordinador y envía un mensaje para informar a todos los sitios activos con números de identificación menores que i de que es el sitio en el que reside el nuevo coordinador.

Si S_i recibe una respuesta, comienza un intervalo de tiempo T' para recibir un mensaje que lo informe de que se ha elegido a un sitio con un número de identificación más elevado (algún otro sitio se está eligiendo coordinador y debe comunicar los resultados dentro del tiempo T'). Si S_i no recibe ningún mensaje antes de T', da por supuesto que el sitio con el número más elevado ha fallado y *Si* vuelve a iniciar el algoritmo.

Después de que se haya recuperado un sitio que ha fallado, comienza de inmediato la ejecución de ese mismo algoritmo. Si no hay ningún sitio activo con un número más elevado, el sitio que se ha recuperado obliga a todos los sitios con números más bajos a permitirle transformarse en el sitio coordinador, aunque ya haya un coordinador activo con un número más bajo. Por este motivo, al algoritmo se le denomina algoritmo *luchador*. Si se divide la red, el alogartimo luchador elige un coordinador separado en cada partición; para asegurar que se elige a lo sumo un coordinador, los sitios ganadores deberían comprobar adicionalmente que una mayoría de los sitios están en su partición.

19.7. PROCESAMIENTO DISTRIBUIDO DE CONSULTAS

En el Capítulo 14 se vio que hay gran variedad de métodos para el cálculo de la respuesta a una consulta. Se examinaron varias técnicas para escoger una estrategia de procesamiento de consultas que minimice la cantidad de tiempo que se tarda en calcular la respuesta. Para los sistemas centralizados el criterio principal para medir el coste de una estrategia dada es el número de accesos a disco. En los sistemas distribuidos hay que tener en cuenta varios asuntos más, entre los que se incluyen

- El coste de la transmisión de los datos por la red
- La ganancia potencial en rendimiento si se hace que varios sitios procesen en paralelo partes de la consulta

El coste relativo de la transferencia de los datos por la red y de la transferencia de los datos al disco o desde él varía ampliamente en función del tipo de red y de la velocidad de los discos. Así, en general, no se puede centrar exclusivamente en los costes de disco ni en los costes de red. Más bien hay que hallar un buen equilibrio entre los dos.

19.7.1. Transformación de consultas

Considérese una consulta extremadamente sencilla: «Hallar todas las tuplas de la relación *cuenta*». Aunque la consulta es sencilla (en realidad, trivial), su procesamiento no es trivial, ya que puede que la relación *cuenta* esté fragmentada, replicada o ambas cosas, como se vio en el Apartado 19.2. Si la relación *cuenta* está replicada, se tiene que elegir la réplica. Si no se han dividido las réplicas, se escoge aquella para la que el coste de transmisión es más bajo. No obstante, si se ha dividido una réplica, la elección no resulta tan sencilla de hacer, dado que hay que calcular varias reuniones o uniones para reconstruir la relación *cuenta*. En ese caso, el número de estrategias para el sencillo ejemplo escogido puede ser grande. Puede que la optimización de las consultas mediante la enumeración exhaustiva de todas las estrategias alternativas no resulte práctica en esas situaciones.

La transparencia de la fragmentación implica que los usuarios pueden escribir una consulta como

$$\sigma_{nombre\text{-}sucursal = \text{«Guadarrama»}} (cuenta)$$

Dado que *cuenta* está definida como

$$cuenta_1 \cup cuenta_2$$

la expresión que resulta del esquema de traducción de nombres es

$$\sigma_{nombre\text{-}sucursal = \text{«Guadarrama»}} (cuenta_1 \cup cuenta_2)$$

Mediante las técnicas de optimización de consultas del Capítulo 13 se puede simplificar de manera automática la expresión precedente. El resultado es la expresión

$$\sigma_{nombre\text{-}sucursal = \text{«Guadarrama»}} (cuenta_1) \cup \sigma_{nombre\text{-}sucursal = \text{«Guadarrama»}} (cuenta_2)$$

que incluye dos subexpresiones. La primera sólo implica a $cuenta_1$y, así, puede evaluarse en el sitio Guadarrama. La segunda sólo implica a $cuenta_2$ y, por tanto, puede evaluarse en el sitio Cercedilla.

Hay otra optimización más que puede hacerse al evaluar

$$\sigma_{nombre\text{-}sucursal = \text{«Guadarrama»}} (cuenta_1)$$

Dado que $cuenta_1$ sólo tiene tuplas pertenecientes a la oficina de Guadarrama, se puede eliminar la operación de selección. Al evaluar

$$\sigma_{nombre\text{-}sucursal = «Guadarrama»} (cuenta_2)$$

se puede aplicar la definición del fragmento de $cuenta_2$ para obtener

$$\sigma_{nombre\text{-}sucursal = «Guadarrama»} (\sigma_{nombre\text{-}sucursal = «Cercedilla»} (cuenta))$$

Esta expresión es el conjunto vacío, independientemente del contenido de la relación *cuenta*.

Por tanto, la estrategia final es que el sitio Guadarrama devuelva $cuenta_1$ como resultado de la consulta.

19.7.2. Procesamiento de reuniones sencillas

Como se vio en el Capítulo 13, una decisión importante en la selección de una estrategia de procesamiento de consultas es la elección de la estrategia de reunión. Considérese la siguiente expresión de álgebra relacional:

$$cuenta \bowtie impositor \bowtie oficina$$

Supóngase que ninguna de las tres relaciones está replicada ni fragmentada, y que *cuenta* está almacenada en el sitio S_1, *impositor* en S_2 y *oficina* en S_3. Supóngase que S_I denota el sitio en el que se ha formulado la consulta. El sistema necesita obtener el resultado en el sitio S_I. Entre las estrategias posibles para el procesamiento de esta consulta figuran las siguientes:

- Enviar copias de las tres relaciones al sitio S_I. Empleando las técnicas del Capítulo 13 hay que escoger una estrategia para el procesamiento local de toda la consulta en el sitio S_I.
- Enviar una copia de la relación *cuenta* al sitio S_2 y calcular $temp_1 = cuenta \bowtie impositor$ en S_2. Enviar $temp_1$ de S_2 a S_3 y calcular $temp_2 = temp_1 \bowtie oficina$ en S_3. Enviar el resultado $temp_2$ a S_I.
- Diseñar estrategias parecidas a la anterior con los roles de S_1, S_2 y S_3 intercambiados.

Ninguna estrategia es siempre la mejor. Entre los factores que deben tenerse en cuenta están el volumen de los datos que se envían, el coste de la transmisión de los bloques de datos entre los pares de sitios y la velocidad relativa de procesamiento en cada sitio. Considérense las dos primeras estrategias mencionadas. Supóngase que los índices presentes de S_2 y S_3 sean útiles para calcular la reunión. Si se envían las tres relaciones a S_I se necesitaría recrear estos índices en S_I o usar una estrategia de reunión diferente y posiblemente más costosa. Esta recreación de los índices supone una sobrecarga adicional de procesamiento y accesos a disco adicionales. Sin embargo, la segunda estrategia tiene el inconveniente de que hay que enviar una relación potencialmente larga (*cuenta* $\bowtie$ *impositor*) de S_2 a S_3. Esta relación repite los datos del nombre de cada cliente una vez por cada cuenta que tenga ese cliente. Así, puede que la segunda estrategia dé lugar a una transmisión de datos adicional por la red en comparación con la primera estrategia.

19.7.3. Estrategia de semirreunión

Supóngase que se desea evaluar la expresión $r_1 \bowtie r_2$, donde r_1 y r_2 se almacenan en los sitios S_1 y S_2, respectivamente. Sean R_1 y R_2 los esquemas de r_1 y de r_2. Supóngase que se desea obtener el resultado en S_1. Si hay muchas tuplas de r_2 que no se reúnan con ninguna tupla de r_1, el envío de r_2 a S_1 supone el envío de las tuplas que no pueden contribuir al resultado. Se desea eliminar esas tuplas antes de enviar los datos a S_1, especialmente si los costes de la red son elevados.

Una estrategia posible para lograr todo esto es la siguiente:

1. Calcular $temp_1 \leftarrow \Pi_{R_1 \cap R_2}(r_1)$ en S_1.
2. Enviar $temp_1$ de S_1 a S_2.
3. Calcular $temp_2 \leftarrow r_2 \bowtie temp_1$ en S_2.
4. Enviar $temp_2$ de S_2 a S_1.
5. Calcular $r_1 \bowtie temp_2$ en S_1. La relación resultante es la misma que $r_1 \bowtie r_2$.

Antes de considerar la eficiencia de esta estrategia hay que verificar que la estrategia calcula la respuesta correcta. En el paso 3 $temp_2$ tiene el resultado de $r_2 \bowtie \Pi_{R_1 \cap R_2}(r_1)$. En el paso 5 se calcula

$$r_1 \bowtie r_2 \bowtie \Pi_{R_1 \cap R_2}(r_1)$$

Dado que la reunión es asociativa y conmutativa, se puede volver a escribir esta expresión como

$$(r_1 \bowtie \Pi_{R_1 \cap R_2}(r_1)) \bowtie r_2$$

Dado que $r_1 \bowtie \Pi_{R_1 \cap R_2}(r_1) = r_1$, la expresión es, realmente, igual a $r_1 \bowtie r_2$, la expresión que se pretendía evaluar.

Esta estrategia resulta especialmente ventajosa cuando relativamente pocas tuplas de r_2 contribuyen a la reunión. Esta situación es probable que se produzca si r_1 es resultado de una expresión de álgebra relacional que implique a una selección. En esos casos puede que $temp_2$ tenga significativamente menos tuplas que r_2. Los ahorros de costes de la estrategia proceden de tener que enviar a S_1 sólo $temp_2$, en vez de toda r_2. El envío de $temp_1$ a S_2 supone un coste adicional. Si una fracción de tuplas de r_2 lo bastante pequeña contribuye a la reunión, la sobrecarga del envío de $temp_1$ queda dominada por el ahorro de tener que enviar sólo una parte de las tuplas de r_2.

Esta estrategia se denomina **estrategia de semirreunión**, del operador de semirreunión del álgebra relacional, denotado por $\ltimes$. La semirreunión de r_1 con r_2, denotada por $r_1 \ltimes r_2$, es

$$\Pi_{R_1}(r_1 \bowtie r_2)$$

Por tanto, $r_1 \ltimes r_2$ selecciona las tuplas de r_1 que han contribuido a $r_1 \bowtie r_2$. En el paso 3 $temp_2 = r_2 \ltimes r_1$.

Para las reuniones de varias relaciones esta estrategia puede ampliarse a una serie de pasos de semirreunión. Se ha desarrollado un importante corpus teórico en relación con el empleo de la semirreunión para la optimización de consultas. A parte de esta teoría se hace referencia en las notas bibliográficas.

19.7.4. Estrategias de reunión que aprovechan el paralelismo

Considérese una reunión de cuatro relaciones:

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

donde la relación r_i se guarda en el sitio S_i. Hay que tener en cuenta que el resultado debe presentarse en el sitio S_1. Hay muchas estrategias posibles para la evaluación en paralelo (el problema del procesamiento de las consultas en paralelo se estudia con detalle en el Capítulo 20). En una de estas estrategias r_1 se envía a S_2 y $r_1 \bowtie r_2$ se calcula en S_2. Al mismo tiempo, r_3 se envía a S_4 y $r_3 \bowtie r_4$ se calcula en S_4. El sitio S_2 puede enviar tuplas de $(r_1 \bowtie r_2)$ a S_1 a medida que se generan, en lugar de esperar a que se calcule toda la reunión. De manera parecida, S_4 puede enviar tuplas de $(r_3 \bowtie r_4)$ a S_1. Una vez que las tuplas de $(r_1 \bowtie r_2)$ y de $(r_3 \bowtie r_4)$ llegan a S_1, puede comenzar el cálculo de $(r_1 \bowtie r_2) \bowtie (r_3 \bowtie r_4)$, con la técnica de reunión canalizada del Apartado 13.7.2.2. Así, el cálculo del resultado de la reunión final en S_1 puede hacerse en paralelo con el cálculo de $(r_1 \bowtie r_2)$ en S_2, y con el de $(r_3 \bowtie r_4)$ en S_4.

19.8. BASES DE DATOS DISTRIBUIDAS HETEROGÉNEAS

Muchas de las últimas aplicaciones de bases de datos necesitan datos de gran variedad de bases de datos existentes previamente y ubicadas en un conjunto heterogéneo de entornos de hardware y de software. El tratamiento de la información ubicada en bases de datos distribuidas heterogéneas exige una capa de software adicional por encima de los sistemas de bases de datos existentes. Esta capa de software se denomina **sistema de varias bases de datos**. Los sistemas locales de bases de datos pueden emplear diferentes modelos lógicos y varios lenguajes de definición y de tratamiento de datos, y puede que se diferencien en sus mecanismos de control de la concurrencia y de administración de las transacciones. Los sistemas de varias bases de datos crean la ilusión de la integración de las bases de datos lógicas sin exigir la integración física de las bases de datos.

La integración completa de sistemas heterogéneos en una base de datos distribuida homogénea suele resultar difícil o imposible:

- **Dificultades técnicas.** La inversión en los programas de aplicaciones basados en los sistemas de bases de datos ya existentes puede ser enorme, y el coste de transformar esas aplicaciones puede resultar prohibitivo.
- **Dificultades organizativas.** Aunque la integración resulte *técnicamente* posible, puede que no lo sea *políticamente*, porque los sistemas de bases de datos ya existentes pertenezcan a diferentes empresas u organizaciones. En ese caso es importante que el sistema de varias bases de datos permita que los sistemas de bases de datos locales conserven un elevado grado de **autonomía** para la base de datos local y para las transacciones que se ejecuten con esos datos.

Por estos motivos los sistemas de varias bases de datos ofrecen ventajas significativas que compensan su sobrecarga. En este apartado se proporciona una introducción a los retos que se afrontan al construir entornos con varias bases de datos desde el punto de vista de la definición de los datos y del procesamiento de las consultas. El Apartado 24.6 ofrece una introducción a los problemas de la administración de las transacciones en varias bases de datos.

19.8.1. Vista unificada de los datos

Cada sistema local de administración de bases de datos puede utilizar un modelo de datos diferente. Por ejemplo, puede que algunos empleen el modelo relacional, mientras que otros pueden emplear modelos de datos más antiguos, como el modelo de red (véase el Apéndice A) o el modelo jerárquico (véase el Apéndice B).

Dado que se supone que los sistemas con varias bases de datos ofrecen la ilusión de un solo sistema de bases de datos integrado, hay que utilizar un modelo de datos común. Una opción adoptada con frecuencia es el modelo relacional, con SQL como lenguaje común de consulta. En realidad hoy en día hay varios sistemas disponibles que permiten realizar consultas SQL en sistemas de administración de bases de datos no relacionales.

Otra dificultad es proporcionar un esquema conceptual común. Cada sistema local ofrece su propio esquema conceptual. El sistema de varias bases de datos debe integrar estos esquemas independientes en un esquema común. La integración de los esquemas es una tarea complicada, sobre todo por la heterogeneidad semántica.

La integración de los esquemas no es meramente la traducción directa de unos lenguajes de definición de datos a otros. Puede que los mismos nombres de atri-

butos aparezcan en bases de datos locales diferentes pero con significados diferentes. Los tipos de datos utilizados en un sistema puede que no estén soportados por otros sistemas y puede que la traducción de unos tipos a otros no resulte sencilla. Incluso en el caso de tipos de datos idénticos pueden surgir problemas debidos a la representación física de los datos: Puede que un sistema utilice ASCII y otro EBCDIC; las representaciones en coma flotante pueden ser diferentes, los enteros pueden representarse como *ordenación natural de bytes* (*big-endian*) o como *ordenación inversa de bytes* (*little-endian*). En el nivel semántico, el valor entero de una longitud pueden ser pulgadas en un sistema y milímetros en otro, lo que crea una situación incómoda en la que la igualdad entre los enteros sea sólo una noción aproximada (como ocurre siempre con los números con coma flotante). Puede que el mismo nombre aparezca en lenguajes distintos en los diferentes sistemas. Por ejemplo, puede que un sistema basado en los Estados Unidos se refiera a la ciudad de Saragossa, mientras que uno con base en España se referirá a ella como Zaragoza.

Todas estas diferencias aparentemente menores deben registrarse de manera adecuada en el esquema conceptual global común. Hay que proporcionar funciones de traducción. Hay que anotar los índices para el comportamiento dependiente del sistema (por ejemplo, el orden de clasificación de los caracteres no alfabéticos no es igual en ASCII que en EBCDIC). Como ya se ha comentado, la alternativa de convertir cada base de datos a un formato común puede que no resulte factible sin dejar obsoletas los programas de aplicación ya existentes.

19.8.2. Procesamiento de las consultas

El procesamiento de las consultas en las bases de datos heterogéneas puede resultar complicado. Algunos de los problemas son:

- Dada una consulta en un esquema global, puede que haya que traducir la consulta a consultas en los esquemas locales de cada uno de los sitios en que hay que ejecutar la consulta. Hay que volver a traducir los resultados de las consultas al esquema global.

  La tarea se simplifica escribiendo **envolturas** para cada origen de datos, que ofrecen una vista de los datos locales en el esquema global. Las envolturas también traducen las consultas del esquema global a consultas del esquema local y vuelven a traducir los resultados al esquema global. Las envolturas puede ofrecerlas cada sitio o escribirse de manera independiente como parte del sistema de varias bases de datos.

  Las envolturas pueden incluso utilizarse para proporcionar una vista relacional de orígenes de datos no relacionales, como las páginas Web (posiblemente con interfaces de formularios), archivos planos, bases de datos jerárquicas y de red y sistemas de directorio.

- Puede que los orígenes de datos sólo ofrezcan posibilidades de consulta limitadas; por ejemplo, puede que soporten las selecciones pero no las reuniones. Puede incluso que restrinjan la forma de las selecciones, permitiéndolas sólo para determinados campos; los orígenes de datos Web con interfaces de formulario son un ejemplo de estos orígenes de datos. Por tanto, puede que haya que dividir las consultas para que se lleven a cabo en parte en el origen de datos y en parte en el sitio que formula la consulta.
- En general, puede que haya que tener acceso a más de un sitio para responder a una consulta dada. Puede que haya que procesar las consultas obtenidas de los diferentes sitios para eliminar los valores duplicados. Supóngase que un sitio contiene las tuplas de *cuenta* que satisfacen las selección *saldo* < 100, mientras que otro contiene las tuplas de *cuenta* que satisfacen *saldo* > 50. Una consulta sobre toda la relación cuenta exigiría tener acceso a los dos sitios y eliminar las respuestas duplicadas consecuencia de las tuplas con saldo entre 50 y 100, que están replicadas en los dos sitios.
- La optimización global de las consultas en bases de datos heterogéneas resulta difícil, ya que puede que el sistema de ejecución de consultas conozca los costes de los planes de consulta alternativos en sitios diferentes. La solución habitual es confiar sólo en la optimización a nivel local y utilizar únicamente la heurística a nivel global.

Los sistemas **mediadores** son sistemas que integran varios orígenes de datos heterogéneos, proporcionan una vista global integrada de los datos y ofrecen facilidades de consulta en el sistema global. A diferencia de los sistemas de varias bases de datos completos, los sistemas mediadores no se ocupan del procesamiento de las transacciones. (Los términos mediador y de varias bases de datos suelen utilizarse de manera indistinta, y puede que los sistemas denominados mediadores soporten formas limitadas de las transacciones.) El término **base de datos virtual** se utiliza para hacer referencia a los sistemas de varias bases de datos o a los sistemas mediadores, ya que ofrecen la apariencia de una sola base de datos con un esquema global, aunque los datos estén en varios sitios en esquemas locales.

19.9. SISTEMAS DE DIRECTORIO

Considérese una organización que desea poner los datos de sus empleados a disposición de diferentes miembros de la organización; entre los datos estarían el nombre, el cargo, el ID de empleado, la dirección, la dirección de correo electrónico, el número de teléfono, el número de fax, etcétera. En los días anteriores a la informática las organizaciones creaban directorios físicos de los empleados y los distribuían por toda la organización. Incluso en nuestros días las compañías telefónicas crean directorios físicos de sus clientes.

En general, un directorio es un listado de la información sobre algunas clases de objetos como las personas. Los directorios pueden utilizarse para hallar información sobre un objeto concreto o, en sentido contrario, hallar objetos que cumplen un determinado requisito. En el mundo de los directorios telefónicos físicos los directorios que permiten las búsquedas en sentido directo se denominan **páginas blancas**, mientras que los directorios que permiten las búsquedas en sentido inverso se denominan **páginas amarillas**.

En el mundo interconectado de hoy en día la necesidad de los directorios sigue vigente y, en todo caso, es aún más importante. No obstante, hoy en día los directorios deben estar disponibles en las redes informáticas en lugar de en forma física (papel).

19.9.1. Protocolos de acceso a directorios

La información de directorio puede dejarse disponible mediante interfaces Web, como hacen muchas organizaciones y, en especial, las compañías telefónicas. Estas interfaces son buenas para las personas que las utilizan. Sin embargo, también los programas necesitan tener acceso a la información de directorio. Los directorios pueden utilizarse para almacenar otros tipos de información, de manera parecida a como hacen los directorios del sistema. Por ejemplo, los exploradores Web pueden almacenar marcas personales de favoritos y otros parámetros del explorador en el sistema de directorios. Por tanto, los usuarios pueden tener acceso a los mismos parámetros desde varias ubicaciones, por ejemplo, desde casa y desde el trabajo, sin tener que compartir un sistema de archivos.

Se han desarrollado varios **protocolos de acceso a directorios** para ofrecer una manera normalizada de acceso a los datos de los directorios. Entre ellos, el más utilizado hoy en día es el **protocolo de acceso ligero a directorios (Lightweight Directory Access Protocol, LDAP**).

Evidentemente, todos los tipos de datos de los ejemplos de este capítulo pueden almacenarse sin demasiados problemas en sistemas de bases de datos y se puede tener acceso a ellos mediante protocolos como JDBC u ODBC. La pregunta, entonces, es el motivo de crear un protocolo especializado para el acceso a la información de directorio. Al menos hay dos respuestas a esta pregunta.

- En primer lugar, los protocolos de acceso a directorios son protocolos simplificados que atienden a un tipo limitado de acceso a los datos. Han evolucionado en paralelo con los protocolos de acceso a las bases de datos.
- En segundo lugar, y lo que es más importante, los sistemas de directorio ofrecen un mecanismo sencillo para nombrar a los objetos de manera jerárquica, parecida a los nombres de directorios de los sistemas de archivos, que pueden utilizarse en un sistema distribuido de directorio para especificar la información que se almacena en cada servidor de directorio. Por ejemplo, puede que un servidor de directorio concreto almacene la información de los empleados de Laboratorios Bell en Cáceres y que otro almacene la información de los empleados de Laboratorios Bell en Zarzalejo, lo que da a ambos sitios autonomía para controlar sus datos locales. Se puede utilizar el protocolo de acceso al directorio para obtener datos de los dos directorios por la red. Lo que es más importante, el sistema de directorios puede configurarse para que envíe de manera automática a un sitio las consultas formuladas en el otro, sin intervención del usuario.

Por estos motivos varias organizaciones tienen sistemas de directorios para hacer que la información de la organización esté disponible en conexión. Como podía esperarse, varias implementaciones de los directorios consideran conveniente utilizar las bases de datos relacionales para almacenar los datos, en lugar de crear sistemas de almacenamiento con finalidad especial.

19.9.2. El protocolo de acceso ligero a directorios LDAP (Lightweight Directory Access Protocol)

En general, los sistemas de directorios se implementan como uno o varios servidores que atienden a varios clientes. Los clientes utilizan la interfaz de programación de aplicaciones definida por el sistema de directorios para comunicarse con los servidores de directorios. Los protocolos de acceso a directorios también definen un modelo de datos y el control de los accesos.

El **protocolo de acceso a directorios X.500**, definido por la organización internacional para la normalización (International Organization for Standardization, ISO), es una norma para el acceso a información de los directorios. No obstante, el protocolo es bastante complejo y no se utiliza demasiado. El **protocolo de acceso ligero a directorios (Lightweight Directory Access Protocol, LDAP**) ofrece muchas de las características de X.500, pero con menos complejidad y se utiliza bas-

tante. En el resto de este apartado se esbozarán detalles del modelo de datos y del protocolo de acceso de LDAP.

19.9.2.1. El modelo de datos LDAP

En LDAP los directorios almacenan **entradas**, que son parecidas a los objetos. Cada entrada debe tener un **nombre distinguido (ND)**, que identifica de manera única esa entrada. Los ND, a su vez, está formado por una secuencia de **nombres distinguidos relativos (NDR)**. Por ejemplo, una entrada puede tener el siguiente nombre distinguido:

cn = Silberschatz, ou = Laboratorios Bell, o = Lucent,
c = USA

Como puede verse, el nombre distinguido de este ejemplo es una combinación de nombre y dirección (dentro de la organización), que comienza por el nombre de la persona y luego da la unidad organizativa (*organizational unit*, ou), organización (*organization*, o) y país (*country*, c). El orden de los componentes del nombre distinguido refleja el orden normal de las direcciones postales en lugar del orden inverso que se utiliza al especificar nombres de caminos para los archivos. El conjunto de NDR de cada ND viene definido por el esquema del sistema de directorio.

Las entradas también pueden tener atributos. LDAP ofrece los tipos binary (binario), string (cadena de caracteres) y time (tiempo) y, de manera adicional, los tipos tel (telefónico) para los números de teléfono y PostalAddress (dirección postal) para las direcciones (las líneas se separan con un carácter «$»). A diferencia de los del modelo relacional, los atributos de manera predeterminada pueden tener varios valores, por lo que es posible almacenar varios números de teléfono o direcciones para una sola entrada.

LDAP permite la definición de **clases de objetos** con nombres y tipos de atributos. Se puede utilizar la herencia para definir las clases de objetos. Además, se pueden especificar que las entradas sean de una clase de objeto o de varias. No es necesario que haya un única clase de objeto más específica a la que pertenezca una entrada dada.

Las entradas se organizan en un **árbol de información del directorio (AID)**, de acuerdo con sus nombres distinguidos. Las entradas en el nivel de las hojas del árbol suelen representar objetos concretos. Las entradas que son nodos internos representan objetos como las unidades organizativas, las organizaciones o los países. Los hijos de cada nodo tienen un ND que contiene todos los NDR del padre, más uno o varios NDR adicionales. Por ejemplo, puede que un nodo interno tenga c=España, y todas las entradas por debajo de él tengan el valor España para el NDR c.

No hace falta almacenar el nombre distinguido completo en una entrada. El sistema puede generar el nombre distinguido de cada entrada recorriendo el AID en sentido ascendente desde la entrada, reuniendo los componentes NDR=valor para crear el nombre distinguido completo.

Puede que las entradas tengan más de un nombre distinguido (por ejemplo, la entrada de una persona en más de una organización). Para tratar estos casos el nivel de las hojas del AID puede ser un **alias**, que apunte a una entrada en otra rama del árbol.

19.9.2.2. Tratamiento de los datos

A diferencia de SQL, LDAP no define ni un lenguaje de definición de datos ni un lenguaje de tratamiento de datos. Sin embargo, LDAP define un protocolo de red para llevar a cabo la definición y el tratamiento de los datos. Los usuarios de LDAP pueden utilizar una interfaz de programación de aplicaciones o las herramientas ofrecidas por varios fabricantes para llevar a cabo la definición y el tratamiento de los datos. LDAP también define un formato de archivos denominado **formato de intercambio de datos LDAP (LDAP Data Interchange Format, LDIF)** que puede utilizarse para almacenar e intercambiar información.

El mecanismo de consulta en LDAP es muy sencillo, consiste simplemente en selecciones y proyecciones, sin ninguna reunión. Cada consulta debe especificar lo siguiente:

- Una base (es decir, un nodo del AID) dando su nombre distinguido (el camino desde la raíz hasta el nodo).
- Una condición de búsqueda, que puede ser una combinación booleana de condiciones para diferentes atributos. Se soportan la igualdad, la coincidencia con caracteres comodín y la igualdad aproximada (la definición exacta de igualdad aproximada depende del sistema).
- Un ámbito, que puede ser sencillamente la base, la base y sus hijos o todo el subárbol por debajo de la base.
- Los atributos que hay que devolver.
- Los límites al número de resultados y al consumo de recursos.

La consulta también puede especificar si hay que eliminar de manera automática las referencias de los alias; si se desactivan las eliminaciones de referencias de los alias se pueden devolver las entradas de los alias como respuestas.

Una manera de consultar orígenes de datos LDAP es emplear los URL LDAP. Ejemplos de los URL LDAP son:

```
ldap::/aura.research.bell-labs.com/o=Lucent,c=USA
ldap::/aura research.bell-labs.com/o=Lucent,c=USA
  ??sub?cn=Korth.
```

El primer URL devuelve todos los atributos de todas las entradas del servidor en que la organización es Lucent y

el país es EEUU. El segundo URL ejecuta una consulta de búsqueda (selección) cn=Korth en el subárbol del nodo con el nombre distinguido o=Lucent, c=USA. Los signos de interrogación de la URL separan campos diferentes. El primer campo es el nombre distinguido, en este caso o=Lucent,c=USA. El segundo campo, la lista de atributos que hay que devolver, se ha dejado vacía, lo que significa que hay que devolver todos los atributos. El tercer atributo, sub, indica que hay que buscar en todo el subárbol. El último parámetro es la condición de búsqueda.

Una segunda manera de consultar un directorio LDAP es utilizar una interfaz de programación de aplicaciones. La Figura 19.6 muestra un fragmento de código C que se utiliza para conectarse con servidores LDAP y ejecutar una consulta en el servidor. El código en primer lugar abre una conexión con un servidor LDAP mediante ldap open y ldap bind. Luego ejecuta una consulta mediante ldap search s. Los argumentos para ldap search s son el controlador de conexión LDAP, el ND de la base desde la que se debe realizar la búsqueda, el ámbito de la búsqueda, la condición de búsqueda, la lista de atributos que hay que devolver y un atributo denominado attrsonly que, si se le asigna el valor de 1, hace que sólo se devuelva el esquema del resultado, sin ninguna tupla real. El último argumento es un argumento de resultados que devuelve el resultado de la búsqueda en forma de estructura LDAPMessage.

El primer bucle **for** itera sobre cada entrada del resultado y la imprime. Obsérvese que cada entrada puede tener varios atributos, y el segundo bucle **for** imprime cada uno de los atributos. Dado que los atributos en LDAP pueden tener varios valores, el tercer bucle **for** imprime cada valor de cada atributo. Las llamadas ldap msgfree y ldap value free liberan la memoria que asignan las bibliotecas LDAP. La Figura 19.6 no muestra el código para tratar las condiciones de error.

La API LDAP también contiene funciones para crear, actualizar y eliminar entradas, así como para otras operaciones con el AID. Cada llamada a una función se comporta como una transacción independiente; LDAP no soporta la atomicidad de las actualizaciones múltiples.

19.9.2.3. Árboles distribuidos de directorio

La información sobre las organizaciones puede hallarse dividida entre varios AIDs, cada uno de los cuales almacena información sobre algunas entradas. El **sufijo** de los AIDs es una secuencia de pares RDN=valores (RDN, Relative Distinguished Name, nombre relativo distinguido) que identifica la información que almacena cada AID; los pares están concatenados con el resto del nombre distinguido generado recorriendo el árbol desde la entrada hasta la raíz. Por ejemplo, el sufijo de un AID puede ser o=Fundent, c=España, mientras que otro puede tener el

```
#include <stdio.h>
#include <ldap.h>
main() { LDAP *ld;
          LDAPMessage *res, *entry;
          char *dn, *attr, *attrList[] = {"telephoneNumber", NULL};
          BerElement *ptr;
          int vals, i;
          ld = ldap_open("aura.research.bell-labs.com", LDAP PORT);
          ldap simple bind(ld, "avi", "avi-passwd") ;
          ldap search s(ld, "o=Lucent, c=USA", LDAP SCOPE SUBTREE, "cn=Korth",
                              attrList, /*attrsonly*/ 0, &res);
          printf("Se han encontrado %d entradas", ldap_count_entries(ld, res));
          for (entry=ldap_first_entry(ld, res); entry != NULL;
                              entry = ldap_next_entry(ld, entry))
          {
                    dn = ldap_get_dn(ld, entry);
                    printf("Nombre distinguido: %s", dn);
                    ldap memfree(dn);
                    for (attr = ldap_first_attribute(ld, entry, &ptr);
                              attr ! NULL;
                              attr = ldap_next_attribute(ld, entry, ptr))
                    {
                              printf("%s: ", attr);
                              vals = ldap_get_values(ld, entry, attr);
                              for (i=0; vals[i] != NULL; i++)
                                        printf("%s, ", vals[i]);
                              ldap value free(vals);
                    }
          }
          ldap_msgfree(res);
          ldap_unbind(ld);
}
```

FIGURA 19.6. Ejemplo de código LDAP en C.

sufijo o=Fundent, c=Chile. Los AIDs pueden estar separados por organizaciones y por criterios geográficos.

Los nodos de un AID pueden contener una **referencia** a un nodo de otro AID; por ejemplo, la unidad organizativa Laboratorios Bell bajo o=Fundent, c=España puede tener su propio AID, en cuyo caso el AID de o=Fundent, c=España tendría el nodo ou=Laboratorios Bell que representará una referencia al AID de Laboratorios Bell.

Las referencias son el componente clave que ayuda a organizar un conjunto distribuido de directorios en un sistema integrado. Cuando un servidor recibe una consulta sobre un AID, puede que devuelva una referencia al cliente, el cual a su vez emite una consulta sobre el AID referenciado. El acceso al AID referenciado es transparente, sin el conocimiento del usuario. Alternativamente, el propio servidor puede emitir la consulta al AID referenciado y devolver los resultados con los resultados calculados localmente.

El mecanismo de denominación jerárquico utilizado por LDAP ayuda a repartir el control de la información entre diferentes partes de la organización. La facilidad de las referencias ayuda a integrar todos los directorios de una organización en un solo directorio virtual.

Aunque no sea un requisito LDAP, las organizaciones suelen escoger repartir la información por criterios geográficos (por ejemplo, una organización puede mantener un directorio por cada sitio en que tenga una presencia importante) o según la estructura organizativa (por ejemplo, cada unidad organizativa, como puede ser un departamento, mantiene su propio directorio).

Muchas implementaciones de LDAP soportan la réplica maestro-esclavo y la réplica multimaestro de los AIDs, aunque la réplica no forme parte de la versión actual de la norma LDAP, la 3. El trabajo de normalización de la réplica en LDAP se halla en curso.

19.10. RESUMEN

- Los sistemas distribuidos de bases de datos consisten en un conjunto de sitios, cada uno de los cuales mantiene un sistema local de bases de datos. Cada sitio puede procesar las transacciones locales: las transacciones que sólo tienen acceso a datos de ese sitio. Además, cada sitio puede participar en la ejecución de transacciones globales, las transacciones que tienen acceso a los datos de varios sitios. La ejecución de las transacciones globales necesita que haya comunicación entre los sitios.
- Las bases de datos distribuidas pueden ser homogéneas, en las que todos los sitios tienen un esquema y un código de sistemas de bases de datos comunes, o heterogéneas, en las que los esquemas y los códigos de los sistemas pueden ser diferentes.
- Hay varios problemas relacionados con el almacenamiento en relación con las bases de datos distribuidas, incluidas la réplica y la fragmentación. Resulta esencial que el sistema minimice el grado en el que los usuarios deben conocer el modo en que se almacenan las relaciones.
- Los sistemas distribuidos pueden sufrir los mismos tipos de fallos que los sistemas centralizados. No obstante, hay fallos adicionales con los que hay que tratar en los entornos distribuidos, entre ellos, el fallo de un sitio, el fallo de un enlace, la pérdida de un mensaje y la división de la red. Cada uno de estos problemas hay que considerarlo en el diseño del esquema distribuido de recuperación.
- Para asegurar la atomicidad todos los sitios en los que se ejecuta la transacción *T* deben estar de acuerdo en el resultado final de su ejecución. O bien *T* se compromete en todos los sitios o se aborta en todos los sitios. Para asegurar esta propiedad el coordinador de la transacción de *T* debe ejecutar un protocolo de compromiso. El protocolo de compromiso más utilizado es el protocolo de compromiso de dos fases.
- El protocolo de compromiso de dos fases puede conducir a bloqueos, la situación en que el destino de una transacción no puede determinarse hasta que se recupere un sitio que ha fallado (el coordinador). Se puede utilizar el protocolo de compromiso de tres fases para reducir la probabilidad de bloqueo.
- La mensajería persistente ofrece un modelo alternativo para el tratamiento de las transacciones distribuidas. El modelo divide cada transacción en varias partes que se ejecutan en bases de datos diferentes. Los mensajes persistentes (que está garantizado que se entregan exactamente una vez, independientemente de los fallos) se envían a los sitios remotos para solicitar que se emprendan acciones en ellos. Aunque la mensajería persistente evita el problema de los bloqueos, los desarrolladores de aplicaciones tienen que escribir código para tratar varios tipos de fallos.
- Los diferentes esquemas de control de la concurrencia empleados en los sistemas centralizados pueden modificarse para su empleo en entornos distribuidos.
  - — En el caso de los protocolos de bloqueo, el único cambio que hay que añadir es el modo en que se implementa el gestor de bloqueos. Hay varios enfoques posibles. Se pueden utilizar uno o varios coordinadores centrales. Si, en vez de eso, se adopta un enfoque con un gestor distribuido de bloqueos, hay que tratar de manera especial los datos replicados.

— Entre los protocolos para el tratamiento de los datos replicados se hallan el protocolo de copia principal, el de mayoría, el sesgado y el de consenso de quórum. Cada uno de ellos tiene diferentes equilibrios en términos de coste y de capacidad de trabajar en presencia de fallos.
— En el caso de los esquemas de marcas temporales y de validación, el único cambio necesario es el desarrollo de un mecanismo para la generación de marcas temporales globales únicas.
— Muchos sistemas de bases de datos soportan la réplica perezosa, en la que las actualizaciones se propagan a las réplicas ubicadas fuera del ámbito de la transacción que llevó a cabo la actualización. Estas facilidades deben utilizarse con grandes precauciones, ya que pueden dar lugar a ejecuciones no secuenciables.

- La detección de interbloqueos en entornos con gestor distribuido de bloqueos exige la colaboración entre varios sitios, dado que puede haber interbloqueos globales aunque no haya interbloqueos locales.
- Para ofrecer una elevada disponibilidad, las bases de datos distribuidas deben detectar los fallos, reconfigurarse de modo que pueda continuar el cálculo y recuperarse cuando se repare el procesador o el enlace. La tarea se complica enormemente por el hecho de que resulta difícil distinguir entre la división de la red y los fallos de los sitios.

  Se puede extender el protocolo de mayoría utilizando números de versiones para permitir que continúe el procesamiento de las transacciones incluso en presencia de fallos. Aunque el protocolo supone una sobrecarga significativa, funciona independientemente del tipo de fallo. Se dispone de protocolos menos costosos para tratar los fallos de los sitios, pero dan por supuesto que no se producen divisiones de la red.
- Algunos algoritmos distribuidos exigen el empleo de coordinadores. Para ofrecer una elevada disponibilidad el sistema debe mantener una copia de seguridad que esté preparada para asumir la responsabilidad si falla el coordinador. Otro enfoque es escoger el nuevo coordinador después de que haya fallado el coordinador. Los algoritmos que determinan el sitio que deberá actuar como coordinador se denominan algoritmos de elección.
- Puede que las consultas a las bases de datos distribuidas necesiten tener acceso a varios sitios. Se dispone de varias técnicas de optimización para escoger los sitios a los que hay que tener acceso. Basadas en la fragmentación y en la réplica, las técnicas pueden utilizar técnicas de semirreunión para reducir la transferencia de datos.
- Las bases de datos distribuidas heterogéneas permiten que cada sitio tenga sus propios esquemas y código de sistema de bases de datos. Los sistemas de varias bases de datos ofrecen un entorno en el que las nuevas aplicaciones de bases de datos pueden tener acceso a los datos de varias bases de datos ya existentes ubicadas en diferentes entornos de hardware y de software heterogéneos. Puede que los sistemas locales de bases de datos empleen modelos lógicos y lenguajes de definición o de manipulación de datos diferentes y puede que se diferencien en los mecanismos de control de la concurrencia o de administración de las transacciones. Los sistemas de varias bases de datos crean la ilusión de la integración lógica de las bases de datos, sin exigir la integración física.
- Los sistemas de directorio pueden considerarse una modalidad especializada de base de datos en la que la información se organiza de manera jerárquica parecida al modo en que los archivos se organizan en los sistemas de archivos. Se tiene acceso a los directorios mediante protocolos normalizados de acceso a directorios como LDAP.

  Los directorios pueden distribuirse entre varios sitios para proporcionar autonomía a cada sitio. Los directorios pueden contener referencias a otros directorios, lo que ayuda a crear vistas integradas en que cada consulta sólo se envía a un directorio y se ejecuta de manera transparente en los directorios correspondientes.

TÉRMINOS DE REPASO

- Algoritmo luchador
- Algoritmos de selección
- Alias
- Árboles distribuidos de directorio
- Autonomía
- Base de datos distribuida heterogénea
- Base de datos distribuida homogénea
- Base de datos virtual
- Control de la concurrencia
- Coordinador suplente
- Coordinador de transacciones
- Copia principal
- Disponibilidad
- División de la red

- Estrategia de semirreunión
- Fragmentación de los datos
  - Fragmentación horizontal
  - Fragmentación vertical
- Gestor distribuido de bloqueos
- Gestor único de bloqueos
- Gestor de transacciones
- Instantánea consistente con las transacciones
- Marcas temporales
- Mediadores
- Mensajería persistente
- Modalidades de fallo del sistema
- Procesamiento distribuido de consultas
- Propagación perezosa
- Protocolo de acceso ligero a directorios LDAP (Lightweight directory access protocol)
  - Árbol de información del directorio (AID)
  - Nombre distinguido (ND)
  - Nombres distinguidos relativos (NDR)
- Protocolo de compromiso de dos fases (C2F)
  - Estado de preparación
  - Problema del bloqueo
  - Transacciones dudosas
- Protocolo de compromiso de tres fases (C3F)
- Protocolos de compromiso
- Protocolos para las réplicas
  - Copia principal
  - Protocolo de consenso de quórum
  - Protocolo de mayoría
  - Protocolo sesgado
- Referencia
- Réplica de datos
- Réplica maestro-esclavo
- Réplica con varios maestros (actualización distribuida)
- Robustez
  - Enfoque basado en la mayoría
  - Leer uno, escribir todo
  - Leer uno, escribir todos los disponibles
  - Reintegración de sitios
- Selección del coordinador
- Servidor de nombres
- Sistema de varias bases de datos
- Sistemas de directorio
- Sufijo AID
- Transacciones distribuidas
  - Transacciones globales
  - Transacciones locales
- Transparencia de los datos
  - Transparencia de la fragmentación
  - Transparencia de la réplica
  - Transparencia de la ubicación
- Tratamiento de los interbloqueos
  - Ciclos falsos
  - Grafo global de espera
  - Grafo local de espera
- Referencia

EJERCICIOS

19.1 Discútanse las ventajas relativas de las bases de datos centralizadas y de las distribuidas.

19.2 Explíquense las diferencias entre transparencia de la fragmentación, transparencia de las réplicas y transparencia de la ubicación.

19.3 Indíquese lo que diferencia a una base de datos distribuida diseñada para una red de área local de otra diseñada para una red de área amplia.

19.4 Indíquese en qué momento resulta útil tener réplicas de los datos o tenerlos fragmentados. Explíquese la respuesta.

19.5 Explíquense los conceptos de transparencia y de autonomía. Indíquese el motivo de que estos conceptos sean deseables desde el punto de vista de los factores humanos.

19.6 Para crear un sistema distribuido con elevada disponibilidad hay que conocer los tipos de fallos que pueden producirse.

  a. Indíquense los tipos de fallos posibles en los sistemas distribuidos.

  b. Indíquense los elementos de la lista de la pregunta a que también sean aplicables a un sistema centralizado.

19.7 Considérese un fallo que se produce durante la ejecución de C2F para una transacción. Para cada fallo posible de los indicados en el Ejercicio 19.6.a explíquese el modo en que C2F asegura la atomicidad de la transacción a pesar del fallo.

19.8 Considérese un sistema distribuido con dos sitios, *A* y *B*. Indíquese si el sitio *A* puede distinguir entre:

- *B* deja de funcionar.
- El enlace entre *A* y *B* deja de funcionar.
- *B* está extremadamente sobrecargado y su tiempo de respuesta es cien veces el habitual.

Indíquense las implicaciones de la respuesta para la recuperación de los sistemas distribuidas.

19.9 El esquema de mensajería persistente descrito en este capítulo depende de las marcas temporales combinadas con el desecho de los mensajes recibidos si son demasiado antiguos. Propóngase un esquema alternativo basado en los números de secuencia en lugar de las marcas temporales.

19.10 Dese un ejemplo en que el enfoque de leer uno, escribir todos los disponibles conduzca a un estado erróneo.

19.11 Si se aplica una versión distribuida del protocolo de granularidad múltiple del Capítulo 16 a una base de datos distribuida, el sitio responsable del DAG puede convertirse en un cuello de botella. Supóngase que se modifica ese protocolo de la manera siguiente:

- Sólo se permiten en la raíz bloqueos en modo tentativo.
- A todas las transacciones se les conceden de manera automática todos los bloqueos en modo tentativo posibles.

Muéstrese que estas modificaciones alivian el problema sin permitir planificaciones no secuenciables.

19.12 Explíquese la diferencia entre la réplica de datos en los sistemas distribuidos y el mantenimiento de sitios remotos de respaldo.

19.13 Dese un ejemplo en el que la réplica perezosa pueda conducir a un estado inconsistente de la base de datos, aunque las actualizaciones obtengan un bloqueo exclusivo sobre la copia principal (maestra).

19.14 Estúdiense y resúmanse las facilidades que el sistema de bases de datos que se está utilizando ofrece para tratar los estados inconsistentes que pueden alcanzarse con la propagación perezosa de las actualizaciones.

19.15 Discútanse las ventajas e inconvenientes de los dos métodos presentados en el Apartado 19.5.2 para la generación de marcas temporales únicas globalmente.

19.16 Considérese el siguiente algoritmo de detección de interbloqueo. Cuando la transacción T_i, en el sitio S_1, solicita un recurso a T_j, en el sitio S_3, se envía un mensaje de solicitud con la marca temporal n. Se inserta el arco (T_i, T_j, n) en el grafo local de espera de S_1. El arco (T_i, T_j, n) sólo se inserta en el grafo local de espera de S_3 si T_j ha recibido el mensaje de solicitud y no se puede conceder de manera inmediata el recurso solicitado. La solicitud de T_i a T_j en el mismo sitio se trata de la manera habitual; no se asocia ninguna marca temporal con el arco (T_i, T_j). El coordinador central invoca el algoritmo de detección enviando el mensaje de inicio a cada sitio del sistema.

Al recibir este mensaje, cada sitio envía al coordinador su grafo local de espera. Obsérvese que ese grafo contiene toda la información local que el sitio tiene sobre el estado del grafo real. El grafo de espera refleja un estado instantáneo del sitio, pero no está sincronizado con respecto a ningún otro sitio.

Cuando el controlador ha recibido una contestación de cada sitio crea un grafo de la manera siguiente:

- El grafo contiene un vértice para cada transacción del sistema.
- El grafo tiene un arco (T_i, T_j) si y sólo si
  - — Hay un arco (T_i, T_j) en uno de los grafos de espera.
  - — Aparece un arco (T_i, T_j, n) (para algún n) en más de un grafo de espera.

Pruébese que, si hay un ciclo en el grafo creado, el sistema se halla en estado de interbloqueo y que, si no hay ningún ciclo en el grafo creado, el sistema no se hallaba en estado de interbloqueo cuando comenzó la ejecución del algoritmo.

19.17 Considérese una relación que está fragmentada horizontalmente por *número-planta*:

empleado (*nombre, dirección, sueldo, número-planta*)

Supóngase que cada fragmento tiene dos réplicas: Una almacenada en el sitio de Madrid y otra almacenada localmente en el sitio de la planta. Descríbase una buena estrategia de procesamiento de las consultas siguientes formuladas en el sitio de Lima.

a. Hallar todos los empleados de la planta de Managua.
b. Hallar el sueldo promedio de todos los empleados.
c. Hallar el empleado mejor pagado de cada uno de los sitios siguientes: Buenos Aires, Rosario, Córdoba, Bahía Blanca.
d. Hallar el empleado peor pagado de la compañía.

19.18 Considérense las relaciones

empleado (*nombre, dirección, sueldo, número-planta*)

máquina (*número-máquina, tipo, número-planta*)

Supóngase que la relación *empleado* está fragmentada horizontalmente por *número-planta* y que cada fragmento se almacena localmente en el sitio de su planta correspondiente. Supóngase que la relación *máquina* se almacena entera en el sitio de Sucre. Descríbase una buena estrategia para el procesamiento de cada una de las consultas siguientes.

a. Hallar todos los empleados de la planta que contiene el número de máquina 1130.
b. Hallar todos los empleados de las plantas que contienen máquinas cuyo tipo sea «trituradora».
c. Hallar todas las máquinas de la planta de Almadén.
d. Hallar empleado $\bowtie$ máquina.

19.19 Para cada una de las estrategias del Ejercicio 19.18 indíquese el modo en que la elección de la estrategia depende:

a. Del sitio en el que se formuló la consulta
b. Del sitio en el que se desea obtener el resultado

20.20 Calcúlese $r \ltimes s$ para las relaciones de la Figura 19.7.

A	B	C
1	2	3
4	5	6
1	2	4
5	3	2
8	9	7

r

C	D	E
3	4	5
3	6	8
2	3	2
1	4	1
1	2	3

s

FIGURA 19.7. Relaciones para el Ejercicio 19.20.

19.21 ¿Es necesariamente $r_i \ltimes r_j$ igual a $r_j \ltimes r_i$? ¿En qué circunstancias es cierto que $r_i \ltimes r_j = r_j \ltimes r_i$?

19.22 Dado que la funcionalidad LDAP puede implementarse sobre un sistema de bases de datos, indíquese la necesidad de la norma LDAP.

19.23 Descríbase el modo en que se puede utilizar LDAP para ofrecer varias vistas jerárquicas de los datos sin replicar los datos del nivel básico.

NOTAS BIBLIOGRÁFICAS

Las discusiones de libro de texto de las bases de datos distribuidas se ofrecen en Ozsu y Valduriez [1999] y en Ceri y Pelagatti [1984]. Las redes de computadoras se discuten en Tanenbaum [1996] y en Halsall [1992]. Rothnie et al. [1977] fue una de las primeras investigaciones sobre sistemas distribuidos de bases de datos. Breitbart et al. [1999b] presenta una introducción a las bases de datos distribuidas.

La implementación del concepto de transacción en bases de datos distribuidas se presenta en Gray [1981], Traiger et al. [1982], Spector y Schwarz [1983] y en Eppinger et al. [1991]. El protocolo C2F lo desarrollaron Lampson y Sturgis [1976] y Gray [1978]. El protocolo de compromiso de tres fases proviene de Skeen [1981]. Mohan y Lindsay [1983] discute dos versiones modificadas de C2F, denominadas *presumir compromiso* y *presumir abortar*, que reducen la sobrecarga de C2F definiendo suposiciones predeterminadas relativas al destino de las transacciones.

El algoritmo luchador del Apartado 19.6.5 proviene de García-Molina [1982]. La sincronización distribuida de los relojes se discute en Lamport [1978]. El control distribuido de la concurrencia se estudia en Rosenkrantz et al. [1978], Bernstein et al. [1978], Bernstein et al. [1980b], Menasce et al. [1980], Bernstein y Goodman [1980], Bernstein y Goodman [1981a], Bernstein y Goodman [1982] y en García-Molina y Wiederhold [1982].

El gestor de transacciones de R* se describe en Mohan et al. [1986]. El control de la concurrencia para los datos replicados que se basa en el concepto de votación se presenta en Gifford [1979] y Thomas [1979]. Las técnicas de validación para el control de los esquemas de concurrencia distribuida se describen en Schlageter [1981], Ceri y Owicki [1983] y Bassiouni [1988]. Las discusiones de las técnicas de administración de las transacciones basadas en la semántica se ofrecen en García-Molina [1983], Kumar y Stonebraker [1988] y Badrinath y Ramamritham [1992].

Attar et al. [1984] discute el empleo de las transacciones en la recuperación distribuida de los sistemas de bases de datos con datos replicados. La investigación de las técnicas para la recuperación en sistemas distribuidos de bases de datos la presenta Kohler [1981].

Recientemente el problema de las actualizaciones concurrentes de los datos replicados ha vuelto a surgir como tema importante de investigación en el contexto de los almacenes de datos. Los problemas en este entorno se discuten en Gray et al. [1996]. Anderson et al. [1998] discute problemas de la réplica perezosa y de la consistencia. Breitbart et al. [1999a] describe los protocolos de actualización perezosa para tratar la réplica. Los manuales de usuario de varios sistemas de bases de datos ofrecen detalles del modo en que tratan la réplica y la consistencia.

La mensajería persistente en Oracle se describe en Gawlick [1998], mientras que Huang y García-Molina [2001] aborda la semántica de sólo-una-vez en los sistemas de mensajería con réplicas.

Los algoritmos de detección distribuida de interbloqueos se presentan en Rosenkrantz et al. [1978], Menasce y Muntz [1979], Gligor y Shattuck [1980], Chandy y Misra [1982], Chandy et al. [1983] y Obermarck [1982]. Knapp [1987] estudia la literatura sobre detección distribuida de interbloqueos, el Ejercicio 19.16 es de Stuart et al. [1984].

El procesamiento distribuido de las consultas se discute en Wong [1977], Epstein et al. [1978], Hevner y Yao [1979], Epstein y Stonebraker [1980], Apers et al. [1983], Ceri y Pelagatti [1983] y Wong [1983]. Selinger y Adiba [1980] y Daniels et al. [1982] discuten el enfoque del procesamiento distribuido de las consultas adoptado por R* (una versión distribuida del Sistema R). Mackert y Lohman [1986] ofrece una evaluación del rendimiento de los algoritmos de procesamiento de las consultas en R*. Los resultados de rendimiento también sirven para validar el modelo de costes utilizado en el optimizador de consultas de R*. Los resultados teóricos referentes a la semirreunión se presentan en Bernstein y Chiu [1981], Chiu y Ho [1980], Bernstein y Goodman [1981b] y Kambayashi et al. [1982].

La optimización dinámica de las consultas en varias bases de datos se aborda en Ozcan et al. [1997]. Adali et al. [1996] y Papakonstantinou et al. [1996] describen los problemas de optimización de las consultas en los sistemas mediadores.

Weltman y Dahbura [2000] y Howes et al. [1999] ofrecen la cobertura de libro de texto de LDAP. Kapitskaia et al. [2000] describe los problemas del almacenamiento en la caché de los datos de directorio de LDAP.

CAPÍTULO

20 BASES DE DATOS PARALELAS

En este capítulo se estudian los algoritmos fundamentales utilizados en los sistemas paralelos de bases de datos basados en el modelo de datos relacional. En concreto, este capítulo se centra en la ubicación de los datos en varios discos y en la evaluación en paralelo de las operaciones relacionales, que han sido esenciales para el éxito de las bases de datos paralelas.

20.1. INTRODUCCIÓN

Hace quince años, los sistemas paralelos de bases de datos han estado casi descartados incluso por algunos de sus más firmes defensores. Actualmente están comercializados con éxito por prácticamente todos las fabricantes de bases de datos. Este cambio lo han impulsado las siguientes tendencias:

- Los requisitos transaccionales de las empresas han aumentado con el uso creciente de las computadoras. Además, el crecimiento de World Wide Web ha creado muchos sitios con millones de visitantes, y las cantidades crecientes de los datos recogidos por los visitantes han producido bases de datos extremadamente grandes en muchas empresas.
- Las empresas utilizan volúmenes crecientes de datos —como los detalles sobre lo que compran las personas, los vínculos que pulsan o la hora a la que realiza las llamadas telefónicas— para planificar sus actividades y sus tarifas. Las consultas utilizadas para estos fines se denominan consultas de **ayuda a la toma de decisiones** y las necesidades de datos para las mismas pueden llegar a los terabytes. Los sistemas con un único procesador no son capaces de tratar volúmenes de datos tan grandes a la velocidad necesaria.
- La naturaleza orientada a conjuntos de las consultas de bases de datos se presta de modo natural a la paralelización. Varios sistemas comerciales y de investigación han demostrado la potencia y dimensionabilidad del procesamiento paralelo de consultas.
- Al abaratarse los microprocesadores, las máquinas paralelas se han vuelto comunes y relativamente baratas.

Como se ha discutido en el Capítulo 18, el paralelismo se utiliza para proporcionar aceleración, y las consultas se ejecutan más rápido debido a que se proporcionan más recursos, como procesadores y discos. El paralelismo también se utiliza para proporcionar ampliabilidad, y las cargas de trabajo crecientes se tratan sin aumentar el tiempo de respuesta mediante un aumento en el grado de paralelismo.

En el Capítulo 18 se esbozaron las diferentes arquitecturas de los sistemas paralelos de bases de datos: de memoria compartida, de discos compartidos, sin compartimiento y las arquitecturas jerárquicas. En resumen, en las arquitecturas de memoria compartida todos los procesadores comparten memoria y discos; en las arquitecturas de disco compartido los procesadores tienen memorias independientes pero comparten los discos; en las arquitecturas sin compartimiento los procesadores no comparten ni la memoria ni los discos; y las arquitecturas jerárquicas tienen nodos que no comparten entre sí ni la memoria ni los discos, pero cada nodo tiene internamente una arquitectura de memoria o de disco compartido.

20.2. PARALELISMO DE E/S

En su forma más sencilla, el **paralelismo de E/S** se refiere a la reducción del tiempo necesario para recuperar relaciones del disco dividiéndolas en varios discos. La forma más frecuente de división de datos en un entorno de bases de datos paralelas es la *división horizontal*. En la **división horizontal**, las tuplas de las relaciones se dividen (o desagrupan) entre varios discos, de modo que cada tupla resida en un disco. Se han propuesto varias estrategias de división.

20.2.1. Técnicas de división

Se presentan tres estrategias básicas para la división de datos. Se da por supuesto que hay n discos, D_0, D_1, ..., D_{n-1}, entre los cuales se van a dividir los datos.

- **Turno rotatorio.** La relación se explora en cualquier orden y la i-ésima tupla se envía al disco numerado $D_{i \bmod n}$. El esquema de turno rotatorio asegura una distribución homogénea de las tuplas entre los discos; es decir, cada disco tiene aproximadamente el mismo número de tuplas que los demás.
- **División por asociación.** En esta estrategia de desagrupación uno o más atributos del esquema de la relación dada se designan como atributos de la división. Se escoge una función de asociación cuyo rango sea {0, 1,..., n-1}. Cada tupla de la relación original se asocia en términos de los atributos de la división. Si la función de asociación devuelve i, la tupla se ubica en el disco D_i.
- **División por rangos.** Esta estrategia distribuye rangos contiguos de valores de los atributos a cada disco. Se escoge un atributo de división, A, como **vector de división.** Sea $[v_0, v_1,..., v_{n-2}]$ el vector de división, tal que , si $i<j$, entonces $v_i<v_j$. La relación se divide como sigue. Considérese una tupla t tal que $t[A]=x$. Si $x<v_0$ entonces t se ubica en el disco D_0. Si $x \geq v_{n-2}$, entonces t se ubica en el disco D_{n-1}. Si $v_i \leq x < v_{i+1}$, entonces t se ubica en el disco D_{i+1}.

Por ejemplo, la división por rangos con tres discos numerados del 0 al 2 puede asignar tuplas con valores menores que 5 al disco 0, con valores entre 5 y 40 al disco 1, y con valores mayores que 40 al disco 2.

20.2.2. Comparación de las técnicas de división

Una vez se ha dividido una relación entre varios discos, se puede recuperar en paralelo utilizándolos todos. De modo parecido, cuando se está dividiendo una relación, se puede escribir en paralelo en varios discos. De esta manera las velocidades de transferencia para la lectura o escritura de una relación completa son mucho mayores con paralelismo de E/S que sin él. Sin embargo, la lectura de una relación completa, o *exploración de la relación* es sólo uno de los tipos de acceso a los datos. El acceso a los datos puede clasificarse de la manera siguiente:

1. Explorar la relación completa.
2. Localizar una tupla de manera asociativa (por ejemplo, *nombre-empleado* = «García»); estas consultas, denominadas **consultas concretas**, buscan tuplas que tengan un valor concreto para un atributo concreto.
3. Localizar todas las tuplas cuyo valor de un atributo dado se halle en un rango especificado (por ejemplo, 10.000<*sueldo*<20.000); estas consultas se denominan **consultas de rangos**.

Las diferentes técnicas de división permiten estos tipos de acceso a diferentes niveles de eficacia:

- **Turno rotatorio.** El esquema se adapta perfectamente a las aplicaciones que desean leer secuencialmente la relación completa para cada consulta. Con este esquema tanto las consultas concretas como las de rangos son difíciles de procesar, dado que se debe emplear en la búsqueda cada uno de los n discos.
- **División por asociación.** Este esquema se adapta mejor a las consultas concretas basadas en el atributo de división. Por ejemplo, si se divide una relación en términos del atributo *número-teléfono*, se puede responder a la consulta «Buscar el registro del empleado con *número-teléfono* = 5553333» aplicando la función de división por asociación a 5553333 y buscando luego en ese disco. Dirigir la consulta a un solo disco ahorra el costo inicial de iniciar una consulta en varios discos y deja a los demás discos libres para procesar otras consultas.

  La división por asociación también es útil para las exploraciones secuenciales de la relación completa. Si la función de asociación es una buena función aleatoria y los atributos de división forman una clave de la relación, el número de tuplas en cada uno de los discos es aproximadamente el mismo, sin mucha varianza. Por tanto, el tiempo empleado para explorar la relación es aproximadamente $1/n$ del tiempo necesario para explorar la relación en un sistema de disco único.

  El esquema, sin embargo, no se adapta bien a las búsquedas concretas en términos de atributos que no sean de división. La división basada en asociación tampoco se adapta bien a las respuestas a consultas de rangos, dado que, generalmente, las funciones de asociación no conservan la proximidad dentro de los rangos. Por tanto, hace falta explorar todos los discos para responder a las consultas de rangos.
- **División por rangos.** Este esquema se adapta bien a las consultas concretas y de rangos basadas en el atributo de división. Para las consultas concretas se puede consultar el vector de división para encontrar el disco en el que reside la tupla. Para las consultas de rangos se consulta el vector de división para hallar el rango de discos en que pueden residir las tuplas. En ambos casos la búsqueda se limita exactamente a aquellos discos que pudieran tener tuplas de interés.

  Una ventaja de esta característica es que, si sólo hay unas pocas tuplas en el rango consultado, la

consulta se suele enviar a un disco, en vez de hacerlo a todos. Dado que se pueden utilizar otros discos para responder a otras consultas, la división por rangos da lugar a una mayor productividad de consultas al tiempo que se mantiene un buen tiempo de respuesta. Por otro lado, si hay muchas tuplas en el rango consultado (como ocurre cuando el rango consultado es una fracción mayor del dominio de la relación), hay que recuperar muchas tuplas de pocos discos, lo que origina un cuello de botella de E/S (punto caliente) en esos discos. En este ejemplo de **sesgo de ejecución** todo el procesamiento tiene lugar en una partición (o en sólo unas pocas). Por el contrario, la división por asociación y de turno rotatorio usarían todos los discos para esas consultas, lo que daría un tiempo de respuesta menor para aproximadamente la misma productividad.

El tipo de división también afecta a otras operaciones relacionales, como las reuniones, tal y como se verá en el Apartado 20.5. De este modo, la elección de la técnica de división también depende de las operaciones que haya que ejecutar. En general, las divisiones por asociación y en rangos se prefieren al turno rotatorio.

En un sistema con muchos discos, el número de discos en los que dividir una relación puede escogerse de la manera siguiente. Si una relación sólo contiene unas pocas tuplas que caben en un solo bloque de disco, es mejor asignar la relación a uno solo. Las relaciones grandes se dividen preferiblemente entre todos los discos disponibles. Si una relación consta de *m* bloques de disco y hay *n* discos disponibles en el sistema, se deberá ubicar la relación en **min**(*m*,*n*) discos.

20.2.3. Tratamiento del sesgo

La distribución de las tuplas al dividir una relación (excepto para el turno rotatorio) puede estar **sesgada**, con un porcentaje alto de tuplas ubicado en algunas divisiones y menos en otros. La manera en que puede aparecer el sesgo se clasifica de la manera siguiente:

- Sesgo de los valores de los atributos.
- Sesgo de la división.

El **sesgo de los valores de los atributos** se refiere al hecho de que algunos valores pueden aparecer en los atributos de división de muchas tuplas. Todas las tuplas con el mismo valor del atributo de división terminan en la misma partición, lo que da lugar al sesgo. El **sesgo de la división** se refiere al hecho de que puede haber un desequilibrio en la carga de la división, aunque no haya sesgo en los atributos.

El sesgo de los valores de los atributos puede dar lugar a una división sesgada independientemente de que se utilice división por rangos o por asociación. Si no se escoge cuidadosamente el vector de división, la división por rangos puede dar lugar a sesgo de división. El sesgo de división es menos probable con división por asociación si se ha escogido una buena función de asociación.

Como se indicó en el Apartado 18.3.1, incluso un sesgo pequeño puede dar lugar a una disminución significativa del rendimiento. El sesgo se transforma en un problema creciente al aumentar el grado de paralelismo. Por ejemplo, si una relación de 1.000 tuplas se divide en 10 partes y la división está sesgada, puede haber algunas particiones de tamaño menor que 100 y otras de tamaño mayor que 100; si incluso da la casualidad de que una partición tiene tamaño 200, la aceleración que se obtendría al tener acceso en paralelo a las particiones es sólo de cinco, en lugar del valor de diez que cabría esperar. Si la misma relación tiene que dividirse en 100 partes, las particiones tendrán de media diez tuplas. Si una partición llega a tener hasta 40 tuplas (lo que es posible dado el gran número de particiones) la aceleración que se obtendría al tener acceso a ellas en paralelo sería de 25, en vez de 100. Por tanto, se puede ver que la pérdida de aceleración debida al sesgo aumenta con el paralelismo.

Se puede construir un **vector de división por rangos equilibrado** mediante ordenación. La relación primero se ordena según los atributos de división. La relación se explora a continuación de forma ordenada. Después de que cada 1/*n* de la relación se haya leído, se añade el valor del atributo de división de la siguiente tupla al vector de división. Aquí, *n* denota el número de particiones a construir. En el caso de que haya el mismo valor para el atributo de división, la técnica puede aún resultar en algo de sesgo. El inconveniente principal de este método es la sobrecarga de E/S debida a la ordenación inicial.

La sobrecarga de E/S debida a la construcción de un vector de división por rangos equilibrado se puede reducir construyendo y almacenado una tabla de frecuencias, o **histograma**, de los valores de atributos para todo atributo de las relaciones. La Figura 20.1 muestra un ejemplo de un histograma para un atributo de tipo entero que

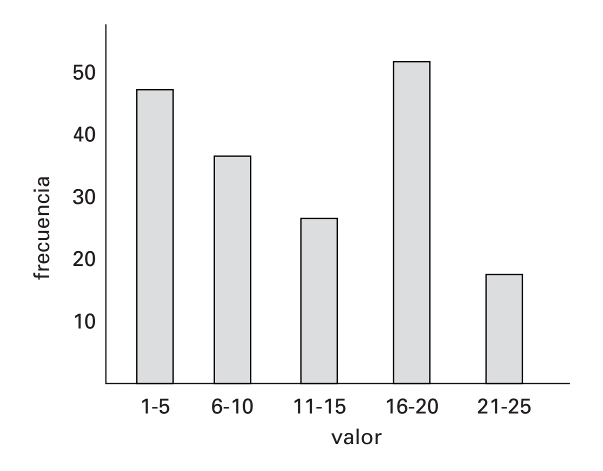


FIGURA 20.1. Ejemplo de histograma.

toma valores en el rango de 1 a 25. Los histogramas ocupan poco espacio, así que en el catálogo se pueden almacenar histogramas sobre diferentes atributos. Es sencillo construir una función de división por rangos equilibrada dado un histograma sobre los atributos de división. Si no se almacena el histograma, se puede calcular de manera aproximada tomando muestras de la relación, usando sólo tuplas de un subconjunto elegido aleatoriamente de los bloques de discos de la relación.

Otro enfoque para minimizar el efecto del sesgo, particularmente con una división por rangos, es usar *procesadores virtuales*. En el enfoque de los **procesadores virtuales**, se supone que hay varias veces procesadores virtuales como procesadores reales haya. Se puede usar cualquiera de las técnicas de división y de evaluación de consultas que se estudian más tarde en este capítulo, pero asignando las tuplas a los procesadores virtuales en lugar de los reales. Los procesadores virtuales, a su vez, se hacen corresponder con procesadores reales, usualmente según una división por turno rotatorio.

La idea es que incluso si un rango tiene muchas más tuplas que los otros debido al sesgo, éstas se puedan repartir entre varios rangos de procesadores virtuales. La asignación por turno rotatorio de los procesadores virtuales a procesadores reales distribuiría el trabajo extra entre varios procesadores reales, de forma que ningún procesador tenga que asumir toda la carga.

20.3. PARALELISMO ENTRE CONSULTAS

En el **paralelismo entre consultas** se ejecutan en paralelo entre sí diferentes consultas o transacciones. La productividad de transacciones puede aumentarse con esta forma de paralelismo. Sin embargo, el tiempo de respuesta de cada transacción no es menor que si éstas se ejecutaran aisladamente. Por ello, el uso principal del paralelismo entre consultas es ampliar los sistemas de procesamiento de transacciones para permitir un número mayor de transacciones por segundo.

El paralelismo entre consultas es la forma más sencilla de paralelismo que se permite en los sistemas de bases de datos, especialmente en sistemas paralelos de memoria compartida. Los sistemas de bases de datos diseñados para sistemas con un único procesador pueden utilizarse en arquitecturas paralelas de memoria compartida con pocos cambios o con ninguno, dado que incluso los sistemas secuenciales de bases de datos permiten el procesamiento concurrente. Las transacciones que se habrían realizado de manera concurrente en tiempo compartido en una máquina secuencial se realizan en paralelo en la arquitectura paralela de memoria compartida.

Permitir el paralelismo entre consultas es más complicado en las arquitecturas de disco compartido y sin compartimiento. Los procesadores tienen que realizar algunas tareas, como los bloqueos y el registro histórico, de forma coordinada, y eso exige que se intercambien mensajes. Los sistemas con arquitectura paralela también deben asegurar que dos procesadores no actualicen simultáneamente los mismos datos de manera independiente. Además, cuando un procesador tiene acceso a los datos o los actualiza, el sistema de bases de datos debe asegurar que el procesador tenga la última versión de éstos en su memoria intermedia. Este último problema se conoce como problema de **coherencia caché**.

Se han desarrollado varios protocolos para garantizar la coherencia caché; a menudo los protocolos de coherencia caché se integran con los de control de la concurrencia de modo que se reduzca la sobrecarga. Un protocolo de este tipo para sistemas de disco compartido es de la manera siguiente:

1. Antes de cualquier acceso de lectura o de escritura a una página, una transacción la bloquea en modo compartido o exclusivo, según corresponda. Inmediatamente después de que la transacción obtenga un bloqueo compartido o exclusivo de la página, lee también la copia más reciente de la misma del disco compartido.
2. Antes de que una transacción libere un bloqueo exclusivo de la página, traslada ésta al disco compartido; luego, libera el bloqueo.

El protocolo anterior asegura que, cuando una transacción establezca un bloqueo compartido o exclusivo sobre una página, obtenga la copia correcta de la página.

Se han desarrollado protocolos más complejos para evitar la lectura y escritura reiteradas del disco exigidas por el protocolo anterior. Con estos protocolos las páginas no se escriben en el disco cuando se liberan los bloqueos exclusivos. Cuando se obtiene un bloqueo compartido o exclusivo, si la versión más reciente de la página está en la memoria intermedia de algún procesador, se obtiene de allí. Los protocolos tienen que diseñarse para tratar peticiones concurrentes. Los protocolos de disco compartido pueden extenderse a las arquitecturas sin compartimiento de la manera siguiente. Cada página tiene un **procesador local** P_i y se guarda en el disco D_i. Cuando otros procesadores quieran leer la página o escribir en ella, enviarán peticiones a su procesador local P_i, dado que no pueden comunicarse directamente con el disco. Las otras acciones son iguales que en los protocolos de disco compartido.

Los sistemas Oracle 8 y Oracle Rdb son ejemplos de sistemas paralelos de bases de datos de disco compartido que permiten paralelismo entre consultas.

20.4. PARALELISMO EN CONSULTAS

El **paralelismo en consultas** se refiere a la ejecución en paralelo de una única consulta en varios procesadores y discos. El uso del paralelismo en consultas es importante para acelerar las consultas de ejecución larga. El paralelismo entre consultas no ayuda en esta labor, dado que cada consulta se ejecuta de manera secuencial.

Para ilustrar la evaluación en paralelo de una consulta considérese una que exija que se ordene una relación. Supóngase que la relación se ha dividido en varios discos mediante la división por rangos basada en algún atributo y que se solicita la ordenación basada en el atributo de división. La operación de ordenación se puede realizar de la manera siguiente: cada partición se ordena en paralelo y las particiones ordenadas se concatenan para obtener la relación ordenada final.

Por tanto, se puede hacer paralela una consulta haciendo paralelas las operaciones que la forman. Hay otra fuente de paralelismo para la evaluación de las consultas: el *árbol de operadores* de una consulta puede contener varias operaciones. Se puede hacer paralela la evaluación del árbol de operadores evaluando en paralelo algunas de las operaciones que no tengan ninguna dependencia entre sí. Además, como se mencionó en el Capítulo 13, puede que se logre encauzar el resultado de una operación hacia otra. Las dos operaciones pueden ejecutarse en paralelo en procesadores separados, uno que genere el resultado que consuma el otro, incluso simultáneamente con su generación.

En resumen, la ejecución de una sola consulta puede hacerse en paralelo de dos maneras:

- **Paralelismo en operaciones.** Se puede acelerar el procesamiento de consultas haciendo paralela la ejecución de cada una de las operaciones, como puede ser la ordenación, la selección, la proyección y la reunión. El paralelismo en operaciones se considera en el Apartado 20.5.
- **Paralelismo entre operaciones.** Se puede acelerar el procesamiento de consultas ejecutando en paralelo las diferentes operaciones de las expresiones de las consultas. Esta forma de paralelismo se considera en el Apartado 20.6.

Las dos formas de paralelismo son complementarias y pueden utilizarse simultáneamente en una misma consulta. Dado que el número de operaciones de una consulta típica es pequeño comparado con el número de tuplas procesado por cada operación, la primera forma de paralelismo puede adaptarse mejor a su aumento. Sin embargo, con el número relativamente pequeño de procesadores de los sistemas paralelos típicos de hoy en día, ambas formas de paralelismo son importantes.

En la discusión siguiente sobre la paralelización en las consultas se da por supuesto que éstas son **sólo de lectura**. La elección de algoritmos para evaluar en paralelo las consultas depende de la arquitectura de la máquina. En lugar de presentar por separado los algoritmos para cada arquitectura se utilizará en la descripción un modelo de arquitectura sin compartimiento. Así, se describirá explícitamente el momento en que los datos deben transferirse de un procesador a otro. Este modelo se puede simular con facilidad utilizando las otras arquitecturas, dado que la transferencia de los datos puede realizarse mediante la memoria compartida en las arquitecturas de memoria compartida y mediante los discos compartidos en las arquitecturas de discos compartidos. Por tanto, los algoritmos para las arquitecturas sin compartimiento también pueden utilizarse en las demás arquitecturas. Ocasionalmente se menciona la manera en que se pueden optimizar aún más los algoritmos para los sistemas de memoria o de discos compartidos.

Para simplificar la exposición de los algoritmos se da por supuesto que hay n procesadores, $P_0, ..., P_{n-1}$ y n discos, $D_0, ..., D_{n-1}$, donde el disco D_i está asociado con el procesador P_i. Un sistema real puede tener varios discos por cada procesador. No es difícil extender los algoritmos para permitir varios discos por procesador: basta con suponer que D_i sea un conjunto de discos. Sin embargo, en aras de la sencillez de la exposición, aquí se supondrá que D_i es un solo disco.

20.5. PARALELISMO EN OPERACIONES

Dado que las operaciones relacionales trabajan con relaciones que contienen grandes conjuntos de tuplas, se pueden paralelizar las operaciones ejecutándolas en paralelo en subconjuntos diferentes de las relaciones. Dado que el número de tuplas de una relación puede ser grande, el grado de paralelismo es potencialmente enorme. Por tanto, el paralelismo en operaciones es natural en los sistemas de bases de datos. Las versiones paralelas de algunas operaciones relacionales frecuentes se estudiarán en los apartados 20.5.1 a 20.5.3.

20.5.1. Ordenación en paralelo

Supóngase que se desea ordenar una relación que reside en n discos, $D_0,..., D_{n-1}$. Si la relación se ha dividido por rangos basándose en los atributos por los que se va

a ordenar, entonces, como se indicó en el Apartado 20.2.2, se puede ordenar por separado cada partición y concatenar los resultados para obtener la relación completa ordenada. Dado que las tuplas se hallan divididas en *n* discos, el tiempo necesario para leer la relación completa se reduce gracias al acceso en paralelo.

Si la relación se ha dividido siguiendo algún otro método se puede ordenar de una de estas dos maneras:

1. Se puede dividir en rangos de acuerdo con los atributos de ordenación y luego ordenar cada partición por separado.
2. Se puede utilizar una versión paralela del algoritmo externo de ordenación-mezcla.

20.5.1.1. Ordenación con división por rangos

La ordenación con división por rangos funciona en dos pasos: primero se divide por rangos la relación y después se ordena cada partición. Cuando se ordena la relación con división por rangos no hace falta dividir en rangos la relación en los mismos procesadores o discos en los que se guarda la relación. Supóngase que se escogen los procesadores P_0, P_1,…,Pm, donde $m<n$ para ordenar la relación. Hay dos pasos involucrados en esta operación:

1. Hay que redistribuir la relación utilizando una estrategia de división por rangos, de manera que todas las tuplas que se hallen dentro del rango *i*-ésimo se envíen al procesador P_i, que guarda temporalmente la relación en el disco D_i.

   Para implementar la división por rangos cada procesador lee en paralelo las tuplas de su disco y envía las tuplas al procesador destino. Cada procesador P_0, P_1,...,Pm también recibe las tuplas de su partición y las almacena localmente. Este paso exige una sobrecarga en la E/S de disco y de comunicaciones.
2. Cada uno de los procesadores ordena localmente su partición de la relación sin interacción con los demás procesadores. Cada procesador ejecuta la misma operación —por ejemplo, ordenar— en un conjunto de datos diferente (la ejecución de la misma acción en paralelo en conjuntos diferentes de datos se denomina **paralelismo de datos**).

   La operación final de mezcla es trivial, ya que la división por rangos de la primera etapa asegura que, para $1 \le i < j \le m$, los valores de la clave del procesador P_i son todos menores que los de P_j.

Hay que dividir por rangos utilizando un buen vector de división por rangos, de manera que cada partición tenga aproximadamente el mismo número de tuplas. Los procesadores virtuales también se pueden utilizar para reducir el sesgo.

20.5.1.2. Ordenación y mezcla externas paralelas

Utilizar la *ordenación y mezcla externas paralelas* es una alternativa a la división por rangos. Supóngase que la relación ya se ha dividido entre los discos D_0, D_1,..., D_{n-1} (no importa la manera en que se haya dividido la relación). La ordenación y la reunión externas paralelas funcionan entonces de la siguiente manera:

1. Cada procesador P_i ordena localmente los datos en el disco D_i.
2. Las partes ordenadas por cada procesador se mezclan luego para obtener el resultado ordenado final.

La mezcla de las partes ordenadas del paso 2 puede hacerse en paralelo de la manera siguiente:

1. Las particiones ordenadas en cada procesador P_i se dividen en rangos (utilizando el mismo vector de división) en los procesadores P_0, P_1,…, P_{m-1}. Las tuplas se envían de acuerdo con el orden de ordenación, por lo que cada procesador recibe las tuplas en corrientes ordenadas.
2. Cada procesador P_i realiza una mezcla de las corrientes según las recibe para obtener una sola parte ordenada.
3. Las partes ordenadas de los procesadores P_0, P_1,…, P_{m-1} se concatenan para obtener el resultado final.

Como se ha descrito, esta secuencia de acciones resulta en una forma interesante del **sesgo de ejecución**, ya que al principio cada procesador envía todos los bloques de la división 0 a P_0, después cada procesador envía todos los bloques de la partición 1 a P_1, y así sucesivamente. Así, mientras que el envío sucede en paralelo, la recepción es secuencial: primero sólo P_0 recibe tuplas, luego sólo P_1 recibe tuplas, y así sucesivamente. Para evitar este problema, cada procesador envía repetidamente un bloque de datos a cada partición. En otras palabras, cada procesador envía el primer bloque de cada partición, después envía el segundo bloque de cada partición, y así sucesivamente. Como resultado, todos los procesadores reciben datos en paralelo.

Algunas máquinas, como las de la serie DBC de Teradata, utilizan hardware especializado para realizar la mezcla. La red de interconexión Y-net de las máquinas DBC de Teradata puede mezclar los resultados de varios procesadores para ofrecer un único resultado ordenado.

20.5.2. Reunión paralela

La operación reunión exige que se comparen pares de tuplas para ver si satisfacen la condición de reunión: si lo hacen, el par se añade al resultado reunido. Los algoritmos de reunión paralela intentan repartir entre varios procesadores los pares que hay que comparar. Cada pro-

cesador procesa luego localmente parte de la reunión. Luego hay que reunir los resultados de cada procesador para producir el resultado final.

20.5.2.1. Reunión por división

Para ciertos tipos de reuniones, como las equirreuniones y las naturales, es posible *dividir* las dos relaciones de entrada entre los procesadores y procesar localmente la reunión en cada uno de ellos. Supóngase que se utilizan n procesadores y que las relaciones que hay que reunir son r y s. Luego, cada una de las relaciones se divide en n particiones, denominadas $r_0, r_1,\ldots, r_{n-1}$ y $s_0, s_1,\ldots, s_{n-1}$. Las particiones r_i y s_i se envían al procesador P_i, donde la reunión se procesa localmente.

La técnica anterior sólo funciona correctamente si la reunión es una equirreunión (por ejemplo, $r \bowtie_{r.A=s.B} s$) y, si se dividen r y s utilizando la misma función de división para sus atributos de reunión. La idea de la división es exactamente la misma que la que subyace al paso de división de la reunión por asociación. Sin embargo, hay dos maneras diferentes de dividir r y s:

- División por rangos de los atributos de reunión
- División por asociación de los atributos de reunión.

En cualquier caso se debe utilizar la misma función de división para las dos relaciones. Para la división por rangos se debe utilizar el mismo vector de división para las dos relaciones. Para la división por asociación esto significa que se debe utilizar la misma función de asociación en las dos relaciones. La Figura 20.2 muestra la división utilizada en una reunión por división paralela.

Una vez divididas las relaciones se puede utilizar localmente cualquier técnica de reunión en cada procesador P_i para calcular la reunión de r_i y s_i. Por ejemplo, se puede utilizar la reunión por asociación, la reunión por mezcla o la reunión con bucles anidados. Por tanto, se puede utilizar la división para paralelizar cualquier técnica de reunión.

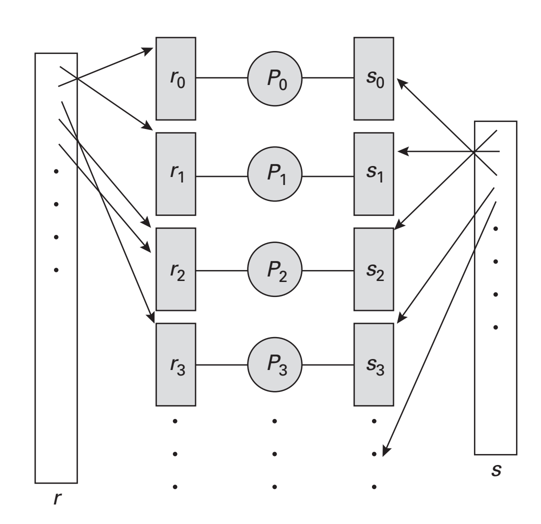


FIGURA 20.2. Reunión por división en paralelo.

Si alguna de las relaciones r y s o ambas ya están divididas basándose en los atributos de reunión (mediante división por asociación o por rangos) el trabajo necesario para la división se reduce mucho. Si las relaciones no están divididas, o lo están basándose en atributos distintos de los de reunión, hay que volver a dividir las tuplas. Cada procesador P_i lee las tuplas del disco D_i, procesa para cada tupla t la partición j a la que pertenece y la envía al procesador P_j. El procesador P_j guarda las tuplas en el disco D_j.

Se puede optimizar el algoritmo de reunión utilizado localmente en cada procesador para reducir E/S no escribiendo en el disco algunas de las tuplas sino guardándolas temporalmente en una memoria intermedia. Estas optimizaciones se describen en el Apartado 20.5.2.3.

El sesgo representa un problema especial cuando se utiliza división por rangos, dado que un vector de división que divida una relación de la reunión en particiones de igual tamaño puede dividir las demás relaciones en particiones de tamaño muy variable. El vector de división debe ser tal que $|r_i| + |s_i|$ (es decir, la suma de los tamaños de r_i y s_i) sea aproximadamente igual para todo $i = 0, 1,\ldots, n-1$. Con una buena función de asociación, la división por asociación probablemente tenga menos sesgo, excepto cuando haya muchas tuplas con los mismos valores de los atributos de reunión.

20.5.2.2. Reunión con fragmentos y réplicas

La división no es aplicable a todos los tipos de reuniones. Por ejemplo, si la condición de reunión es una desigualdad, como $r \bowtie_{r.a<s.b} s$, es posible que todas las tuplas de r se reúnan con alguna tupla de s (y viceversa). Por tanto, puede que no haya medios no triviales de dividir r y s de modo que las tuplas de la partición r_i se reúnan exclusivamente con tuplas de la partición s_i.

Esas reuniones pueden paralelizarse utilizando una técnica denominada *fragmentos y réplicas*. Primero se considerará un caso especial de fragmentos y réplicas —**reunión con fragmentos y réplicas asimétricos**— que funciona de la manera siguiente.

1. Se divide una de las relaciones (digamos r). Se puede utilizar en r cualquier técnica de división, incluyendo la división por turno rotatorio.
2. La otra relación, s, se replica entonces en todos los procesadores.
3. El procesador P_i procesa entonces localmente la reunión de r_i con toda s, utilizando cualquier técnica de reunión.

El esquema de fragmentos y réplicas asimétricos se muestra en la Figura 20.3a. Si r ya está guardada por particiones no hace falta dividirla más en el paso 1. Todo lo que hace falta es replicar s en todos los procesadores.

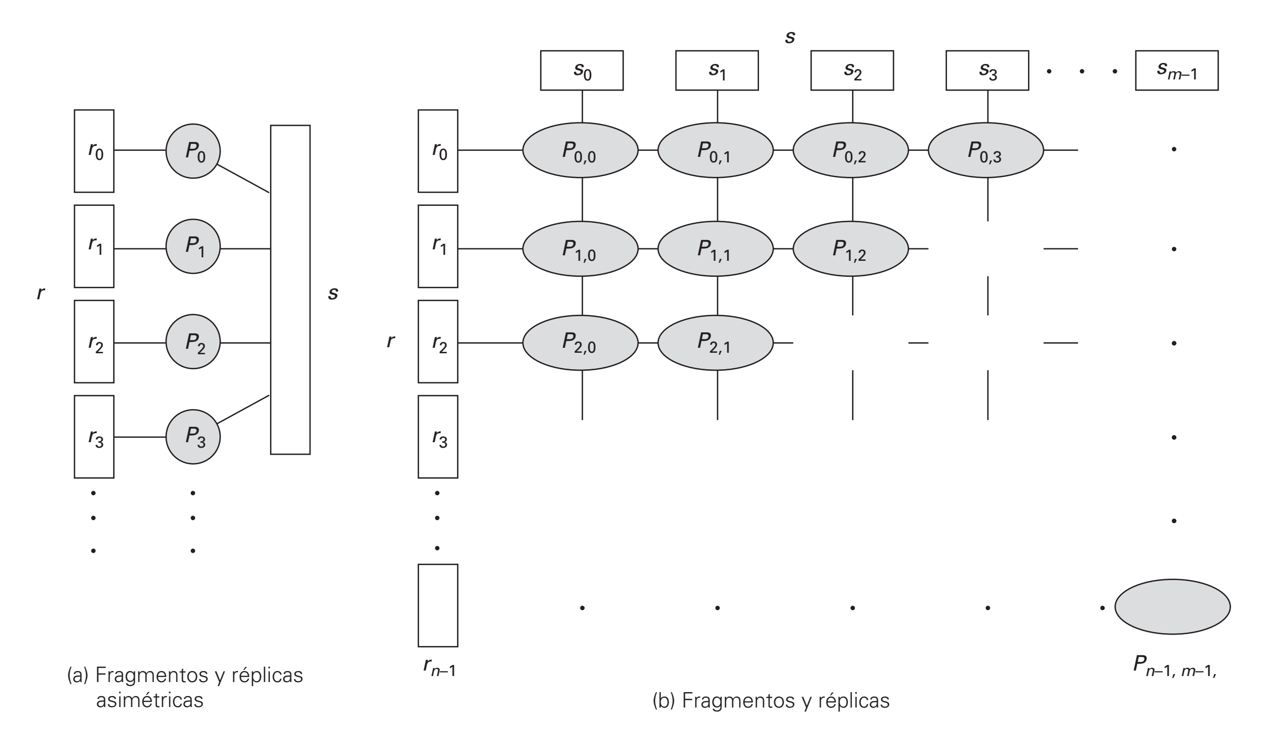


FIGURA 20.3. Esquemas de fragmentos y réplicas.

El caso general de **reunión con fragmentos y réplicas** se muestra en la Figura 20.3b; funciona de la manera siguiente. La relación r se divide en n particiones, r_0, $r_1, \ldots, r_{n-1}$ y s se divide en m particiones, $s_0, s_1, \ldots, s_{m-1}$. Igual que antes se puede utilizar cualquier técnica de división para r y para s. Los valores de m y de n no tienen por qué ser iguales, pero deben escogerse de modo que haya al menos $m * n$ procesadores. Los fragmentos y réplicas asimétricos sólo son un caso especial de los fragmentos y réplicas generales, donde $m = 1$. Los fragmentos y réplicas reducen el tamaño de las relaciones en cada procesador en comparación con los fragmentos y réplicas asimétricos.

Sean los procesadores $P_{0,0}, P_{0,1}, \ldots, P_{0,m-1}, P_{1,0}, \ldots, P_{n-1,m-1}$. El procesador $P_{i,j}$ procesa la reunión de r_i con s_j. Para ello se replica r_i en los procesadores $P_{i,0}, P_{i,1}, \ldots, P_{i,m-1}$ (que forman una fila en la Figura 20.3b) y s_i en los procesadores $P_{0,i}, P_{1,i}, \ldots, P_{n-1,i}$ (que forman una columna en la Figura 20.3b). Se puede utilizar cualquier técnica de reunión en cada procesador $P_{i,j}$.

Los fragmentos y réplicas funcionan con cualquier condición de reunión, dado que todas las tuplas de r pueden compararse con todas las tuplas de s. Por tanto, puede utilizarse cuando no pueda emplearse la división.

Los fragmentos y réplicas suelen tener un mayor coste que la división cuando ambas relaciones son aproximadamente del mismo tamaño, dado que hay que replicar al menos una de las relaciones. Sin embargo, si una de las relaciones (digamos s) es pequeña, puede resultar más barato replicar s en todos los procesadores que volver a dividir r y s basándose en los atributos de reunión. En tal caso, los fragmentos y réplicas asimétricos son preferibles, aunque pueda utilizarse la división.

20.5.2.3. Reunión por asociación dividida en paralelo

En este apartado se muestra la manera en que la reunión por asociación dividida del Apartado 13.5.5 puede hacerse en paralelo. Supóngase que se tienen n procesadores, $P_0, P_1, \ldots, P_{n-1}$ y dos relaciones, r y s que se encuentran divididas en varios discos. Hay que recordar del Apartado 12.6 que se debe escoger la relación menor como relación de construcción. Si el tamaño de s es menor que el de r, el algoritmo de reunión por asociación procede de la manera siguiente:

1. Se escoge una función de asociación (digamos h_1) que tome el valor del atributo de reunión de cada tupla y asigne esta tupla a uno de los n procesadores. Sean r_i las tuplas de la relación r que se envían al procesador P_i; de manera similar, sean s_i las tuplas de la relación s que se envían al procesador P_i. Cada procesador P_i lee las tuplas de s que están en el disco D_i y envía cada tupla al procesador apropiado basándose en la función de asociación h_1.
2. Según se reciben en el procesador de destino P_i las tuplas de s_i, se vuelven a dividir utilizando otra función de asociación, h_2, que el procesador utiliza para procesar localmente la reunión por asociación. La división en esta etapa es exactamente la misma que en la fase de división del

algoritmo secuencial de reunión por asociación. Cada procesador P_i ejecuta este paso independientemente de los demás procesadores.

3. Una vez que se han distribuido las tuplas de s, el sistema redistribuye la relación mayor, r, entre los n procesadores utilizando la función de asociación h_1 del mismo modo que anteriormente. Según se recibe cada tupla, el procesador de destino la divide utilizando la función h_2, igual que la relación de prueba se divide en el algoritmo secuencial de reunión por asociación.
4. Cada procesador P_i ejecuta las fases de construcción y prueba del algoritmo de reunión por asociación en las particiones locales r_i y s_i para producir una división del resultado final de la reunión por asociación.

La reunión por asociación realizada en cada procesador es independiente de las realizadas en otros procesadores, y recibir las tuplas de r_i y de s_i es parecido a leerlas del disco. Por tanto, cualquiera de las optimizaciones de la reunión por asociación descritas en el Capítulo 13 pueden aplicarse también al caso paralelo. En concreto, se puede utilizar el algoritmo híbrido de reunión por asociación para guardar en caché algunas de las tuplas de entrada, y evitar así los costes de escribirlas y volver a leerlas.

20.5.2.4. Reuniones con bucles anidados en paralelo

Para ilustrar el uso de la paralelización basada en fragmentos y réplicas se considera el caso en que la relación s sea mucho menor que la relación r. Supóngase también que la relación r se guarda dividiéndola; el atributo según el que se divide es irrelevante. Finalmente, supóngase que hay un índice basado en un atributo de reunión de la relación r en cada uno de las particiones de la relación r.

Se utilizan los fragmentos y réplicas asimétricos, mientras se replica la relación s y se utiliza la división existente de la relación r. Cada procesador P_j en que se guarda una partición de la relación s lee las tuplas de la relación s guardadas en D_j y replica las tuplas en el resto de procesadores P_i. Al final de esta fase la relación s está replicada en todos los puntos que guardan tuplas de la relación r.

Ahora, cada procesador P_i realiza una reunión con bucles anidados indexada de la relación s con la partición i-ésima de la relación r. Se puede solapar la reunión con bucles anidados indexada con la distribución de las tuplas de la relación s para reducir los costes de escribir en el disco las tuplas de la relación s y de volver a leerlas. Sin embargo, la réplica de la relación s debe sincronizarse con la reunión para que haya espacio suficiente en las memorias intermedias de la memoria principal de cada procesador P_i para guardar las tuplas de la relación s que se hayan recibido pero que todavía no se han utilizado en la reunión.

20.5.3. Otras operaciones relacionales

También se puede realizar en paralelo la evaluación de otras operaciones relacionales:

- **Selección.** Sea la selección $\sigma_\theta(r)$. Considérese primero el caso en el que θ es de la forma $a_i = v$, donde a_i es un atributo y v es un valor. Si la relación r se divide basándose en a_i la selección se lleva a cabo en un solo procesador. Si θ es de la forma $l \leq a_i \leq u$ (es decir, que θ es una selección de rango) y la relación se ha dividido por rangos basándose en a_i, entonces la selección se lleva a cabo en cada procesador cuya partición se solape con el rango de valores especificado. En el resto de los casos la selección se lleva a cabo en todos los procesadores en paralelo.
- **Eliminación de duplicados.** La eliminación de duplicados puede llevarse a cabo por ordenación; puede utilizarse cualquiera de las técnicas de ordenación en paralelo, con la optimización de eliminar los duplicados durante la ordenación tan pronto como se encuentren. También se puede realizar la eliminación de duplicados en paralelo dividiendo las tuplas (mediante división por rangos o por asociación) y llevando a cabo localmente en cada procesador la eliminación de duplicados.
- **Proyección.** Se puede llevar a cabo la proyección sin eliminación de duplicados según se leen en paralelo las tuplas del disco. Si se va a llevar a cabo la eliminación de duplicados se pueden utilizar las técnicas que se acaban de describir.
- **Agregación.** La agregación puede considerarse una operación. Se puede paralelizar la operación dividiendo la relación basándose en los atributos de agrupación y procesando luego localmente los valores de agregación en cada procesador. Se puede utilizar división por rangos o por asociación. Si la relación ya está dividida basándose en los atributos de agrupación se puede omitir el primer paso.

  Se puede reducir el coste de transferir las tuplas durante la división procesando parcialmente los valores de agregación antes de la división, al menos con las funciones de agregación utilizadas frecuentemente. Considérese una operación de agregación basada en una relación r, utilizando la función de agregación **sum** con el atributo B y la agrupación basada en el atributo A. La operación puede llevarse a cabo en cada procesador P_i sobre las r tuplas guardadas en el disco D_i. Este proceso da lugar a tuplas con sumas parciales en cada procesador; hay una tupla en P_i para cada valor del atributo A presente en r tuplas guardadas en D_i. El resultado de la agregación local se divide basándose en el atributo de agrupación A y la agregación se vuelve a llevar a cabo (sobre las tuplas con sumas parciales) en cada procesador P_i para obtener el resultado final.

Como resultado de esta optimización no es necesario enviar tantas tuplas a los demás procesadores durante la división. Esta idea puede extenderse fácilmente a las funciones de agregación **min** y **max**. Las extensiones para las funciones de agregación **count** y **avg** se proponen al lector en el Ejercicio 20.8.

La paralelización de otras operaciones se trata en varios ejercicios.

20.5.4. Coste de la evaluación en paralelo de las operaciones

Se puede obtener el paralelismo dividiendo la E/S entre varios discos y el trabajo de la UCP entre varios procesadores. Si se logra un reparto así sin sobrecarga y no hay sesgo en el reparto del trabajo, las operaciones en paralelo que utilicen *n* procesadores tardarán *1/n* lo que tardarían en un solo procesador. Ya se sabe cómo estimar el coste de operaciones como la reunión o la selección. El coste en tiempo del procesamiento paralelo sería entonces *1/n* el del procesamiento secuencial de la operación.

También hay que tener en cuenta los costes siguientes:

- Los **costes de iniciar** la operación en varios procesadores.
- El **sesgo** en la distribución de trabajo entre los procesadores, con algunos procesadores con mayor número de tuplas que otros.
- La **contención de recursos** —como la memoria, los discos y la red de comunicaciones— que dan lugar a retrasos.
- El **coste de construir** el resultado final transmitiendo los resultados parciales desde cada procesador.

El tiempo empleado por una operación en paralelo puede estimarse como

$$T_{\text{div}} + T_{\text{con}} + \max(T_0, T_1, \ldots, T_{n-1})$$

Donde T_{div} es el tiempo necesario para dividir las relaciones, T_{con} es el tiempo empleado en construir los resultados y T_i el tiempo utilizado por la operación en el procesador P_i. Suponiendo que las tuplas se distribuyen sin sesgo, el número de tuplas enviadas a cada procesador puede estimarse como $1/n$ del número total de tuplas. Ignorando la contención, el coste T_i de las operaciones en cada procesador P_i, puede estimarse entonces mediante las técnicas descritas en el Capítulo 13.

La estimación precedente será una estimación optimista, dado que el sesgo es frecuente. Aunque dividir una sola consulta en varios pasos en paralelo reduce el tamaño del paso medio, es el tiempo de procesamiento del paso más lento el que determina el tiempo empleado en procesar la consulta en su conjunto. Una evaluación en paralelo de las particiones, por ejemplo, sólo es tan rápida como la más lenta de sus ejecuciones en paralelo. Por tanto, el rendimiento se ve muy afectado por cualquier sesgo en la distribución del trabajo entre los procesadores.

El problema del sesgo de la división está íntimamente relacionado con el del desbordamiento de particiones en las reuniones por asociación secuenciales (Capítulo 13). Se puede utilizar la resolución del desbordamiento y las técnicas de evitación desarrolladas para las reuniones por asociación para tratar el sesgo cuando se utilice la división por asociación.

20.6. PARALELISMO ENTRE OPERACIONES

Hay dos formas de paralelismo entre operaciones: el paralelismo de encauzamiento y el paralelismo independiente.

20.6.1. Paralelismo de encauzamiento

Como se discutió en el Capítulo 13, el encauzamiento supone una importante fuente de economía de procesamiento para el procesamiento de consultas de bases de datos. Hay que recordar que, en el encauzamiento, las tuplas resultado de una operación, *A*, las consume una segunda operación, *B*, incluso antes de que la primera operación haya producido el conjunto completo de tuplas de su resultado. La ventaja principal de la ejecución encauzada de las evaluaciones secuenciales es que se puede ejecutar una secuencia de operaciones de ese tipo sin escribir en el disco ninguno de los resultados intermedios.

En los sistemas paralelos el encauzamiento se utiliza principalmente por la misma razón que en los sistemas secuenciales. Sin embargo, el encauzamiento puede utilizarse también como fuente de paralelismo, del mismo modo que el encauzamiento de instrucciones se utiliza como fuente de paralelismo en el diseño de hardware. En el ejemplo anterior es posible ejecutar simultáneamente *A* y *B* en procesadores diferentes de modo que *B* consuma las tuplas en paralelo con su producción por *A*. Esta forma de paralelismo se denomina **paralelismo de encauzamiento**.

Consideremos una reunión de cuatro relaciones:

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

Se puede configurar un cauce que permita que las tres reuniones se procesen en paralelo. Supóngase que se asigna al procesador P_1 el cálculo de $temp_1 \leftarrow r_{10} \bowtie$

r_2 y al procesador P_2 el cálculo de $r_3 \bowtie$ *temp*$_1$. Mientras P_1 procesa las tuplas de $r_1 \bowtie r_2$ las pone a disposición del procesador P_2. Por tanto, P_2 tiene a su disposición algunas de las tuplas de $r_1 \bowtie r_2$ antes de que P_1 haya finalizado su cálculo. P_2 puede utilizar esas tuplas que están disponibles para comenzar el cálculo de *temp*$_1 \bowtie r_3$, incluso antes de que P_1 haya procesado completamente $r_1 \bowtie r_2$. De manera parecida, mientras P_2 procesa las tuplas de $(r_1 \bowtie r_2) \bowtie r_3$, pone estas tuplas a disposición de P_3, que procesa la reunión de estas tuplas con r_4.

El paralelismo encauzado resulta útil con un número pequeño de procesadores, pero no puede extenderse bien. En primer lugar, las cadenas del cauce no suelen lograr la longitud suficiente para proporcionar un alto grado de paralelismo. En segundo lugar, no es posible encauzar los operadores de relación que no producen resultados hasta que se ha tenido acceso a todas las entradas, como la operación diferencia de conjuntos. En tercer lugar, sólo se obtiene una aceleración marginal en los casos frecuentes en que el coste de ejecución de un operador es mucho mayor que los de los demás operadores.

Por consiguiente, cuando el grado de paralelismo es elevado, la importancia del encauzamiento como fuente de paralelismo es secundaria respecto de la del paralelismo de particiones. La razón fundamental para utilizar el encauzamiento es que las ejecuciones encauzadas pueden evitar escribir en el disco los resultados intermedios.

20.6.2. Paralelismo independiente

Las operaciones en las expresiones de las consultas que son independientes entre sí pueden ejecutarse en paralelo. Esta forma de paralelismo se denomina **paralelismo independiente**.

Considérese nuevamente la reunión $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$. Claramente, se puede procesar *temp*$_1 \leftarrow r_1 \bowtie r_2$ en paralelo con *temp*$_2 \leftarrow r_3 \bowtie r_4$. Cuando se completen estos dos cálculos se calculará

$$temp_1 \bowtie temp_2$$

Para obtener más paralelismo se pueden encauzar las tuplas de *temp*$_1$ y de *temp*$_2$ al cálculo de *temp*$_1 \bowtie$ *temp*$_2$, que se ejecuta mediante una reunión encauzada (Apartado 13.7.2.2).

Al igual que el paralelismo encauzado, el paralelismo independiente no proporciona un alto grado de paralelismo y es menos útil en sistemas con un elevado nivel de paralelismo, aunque resulta útil con un grado menor de paralelismo.

20.6.3. Optimización de consultas

Un factor importante en el éxito de la tecnología relacional ha sido el diseño con éxito de optimizadores de consultas. Recuérdese que un optimizador de consultas toma una consulta y encuentra el plan de ejecución más económico de entre los muchos planes de ejecución posibles que proporcionan la misma respuesta.

Los optimizadores de consultas para la evaluación de consultas en paralelo son más complicados que los optimizadores de consultas para la evaluación secuencial de consultas. En primer lugar, los modelos de costes son más complicados dado que hay que tener en cuenta los costes de división, y deben tenerse en consideración aspectos como el sesgo y la contención de recursos. Resulta de mayor importancia el asunto de la paralelización de las consultas. Supóngase que de algún modo se ha escogido una expresión (de entre las equivalentes a la consulta) para utilizarla para evaluar la consulta. La expresión puede representarse por un árbol de operadores, tal y como se discute en el Apartado 13.1.

Para evaluar un árbol de operadores en un sistema paralelo hay que tomar las decisiones siguientes:

- El modo en que se paralelice cada operación y el número de procesadores que se utilizará para ello.
- Las operaciones que se encauzan entre los diferentes procesadores, las operaciones que se ejecuten independientemente en paralelo y las que se ejecuten secuencialmente, una tras otra.

Estas decisiones constituyen la tarea de **planificar** el árbol de ejecución.

Determinar los recursos de cada clase —como procesadores, discos y memoria— que se deben asignar a cada operación del árbol es otro aspecto del problema de la optimización. Por ejemplo, puede que parezca conveniente utilizar la máxima cantidad disponible de paralelismo, pero es una buena idea no ejecutar en paralelo ciertas operaciones. Las operaciones cuyos requisitos operativos sean significativamente menores que la sobrecarga de comunicaciones deben agruparse con una de sus vecinas. En caso contrario, la ventaja del paralelismo se anula por sobrecarga en las comunicaciones.

Una preocupación cuando se utiliza el encauzamiento es que los cauces largos no se presten a una buena utilización de los recursos. A menos que las operaciones tengan grano grueso, puede que la operación final del encauzamiento espere mucho tiempo para obtener sus entradas, mientras retiene recursos preciosos, como la memoria. Por tanto, deben evitarse los cauces largos.

El número de planes de evaluación en paralelo de entre los que se puede escoger es mucho mayor que el de planes de evaluación secuenciales. Optimizar las consultas en paralelo considerando todas las alternativas es por tanto mucho más costoso que optimizar las consultas secuenciales. Por tanto, se suelen adoptar enfoques heurísticos para reducir el número de planes de ejecución en paralelo que se considera. A continuación se describen dos heurísticas muy conocidas.

La primera heurística es considerar únicamente los planes de evaluación que paralelizan todas las operacio-

nes de todos los procesadores y no utilizan encauzamiento. Este enfoque se utiliza en las máquinas de la serie DBC de Teradata. Buscar el mejor plan de ejecución de este tipo es parecido a realizar la optimización de consultas en sistemas secuenciales. Las principales diferencias radican en la manera de llevar a cabo la división y en las fórmulas de estimación de costes utilizadas.

La segunda heurística es escoger el plan de evaluación secuencial más eficiente y luego paralelizar las operaciones de ese plan de evaluación. La base de datos paralela Volcano ha popularizado el modelo de paralelización denominado **intercambio de operadores**. Este modelo usa implementaciones existentes de operaciones, actuando sobre copias locales de los datos, junto con una operación de intercambio que traslada los datos entre diferentes procesadores. Los operadores de intercambio se pueden introducir en un plan de evaluación para transformarlo en un plan de evaluación en paralelo.

Otra dimensión más de la optimización es el diseño de la organización del almacenamiento físico para acelerar las consultas. La organización física óptima es diferente para consultas diferentes. El administrador de la base de datos debe escoger una organización física que se considere adecuada para la combinación esperada de consultas a la base de datos.

Por tanto, el área de la optimización de consultas en paralelo es compleja y sigue siendo un campo de investigación activa.

20.7. DISEÑO DE SISTEMAS PARALELOS

Hasta ahora, en este capítulo hemos centrado la atención en la paralelización del almacenamiento de los datos y en el procesamiento de consultas. Dado que los sistemas paralelos de bases de datos de gran escala se utilizan principalmente para almacenar grandes volúmenes de datos y para procesar consultas de ayuda a las decisiones basadas en dichos datos, estos temas son los más importantes en los sistemas paralelos de bases de datos. La carga de datos en paralelo desde fuentes externas es un requisito importante si se van a tratar grandes volúmenes de datos entrantes.

Un gran sistema paralelo de bases de datos debe abordar también los siguientes aspectos de disponibilidad:

- El poder de recuperación frente al fallo de algunos procesadores o discos
- La reorganización interactiva de los datos y los cambios de los esquemas.

Estos temas se tratan a continuación.

Con un gran número de procesadores y de discos la probabilidad de que al menos un procesador o un disco funcionen mal es significativamente mayor que en sistema con un único procesador y un solo disco. Un sistema paralelo mal diseñado dejará de funcionar si cualquier componente (procesador o disco) falla. Suponiendo que la probabilidad de fallo de cada procesador o disco es pequeña, la probabilidad de fallo del sistema asciende linealmente con el número de procesadores y de discos. Si un solo procesador o disco falla una vez cada 5 años, un sistema con 100 procesadores tendrá un fallo cada 18 días.

Por tanto, los sistemas paralelos de bases de datos de gran escala, como las máquinas Himalaya de Compaq, Teradata y XPS de Informix (ahora una división de IBM), se diseñan para operar incluso si falla un procesador o un disco. Los datos se replican en al menos dos procesadores. Si falla un procesador se puede seguir teniendo acceso desde los demás procesadores a los datos que guarda. El sistema hace un seguimiento de los procesadores con fallos y distribuye el trabajo entre los que funcionan. Las peticiones de los datos guardados en el emplazamiento con fallo se desvían automáticamente a los emplazamientos de las copias de seguridad que guardan una réplica de los datos. Si todos los datos de un procesador *A* se replican en un solo procesador *B*, *B* tratará todas las peticiones hechas a *A*, así como las propias, y eso dará lugar a que *B* se transforme en un cuello de botella. Por tanto, las réplicas de los datos de un procesador se dividen entre varios procesadores.

Cuando se manejan grandes volúmenes de datos (del orden de terabytes), las operaciones sencillas, como la creación de índices, y los cambios en los esquemas, como añadir una columna a una relación, pueden tardar mucho tiempo (quizás horas o incluso días). Por tanto, es inaceptable que los sistemas de bases de datos no estén disponibles mientras se llevan a cabo tales operaciones. Los sistemas paralelos de bases de datos, como los sistemas Himalaya de Compaq, permiten que tales operaciones se lleven a cabo **interactivamente**, es decir, mientras el sistema ejecuta otras transacciones.

Considérese, por ejemplo, la **construcción interactiva de índices**. Un sistema que permita esta prestación permite que se realicen inserciones, borrados y actualizaciones en una relación aunque se esté generando un índice de la misma. La operación de generación de índices, por tanto, no puede bloquear toda la relación en modo compartido, como habría hecho en caso contrario. En su lugar, el proceso hace un seguimiento de las actualizaciones que tienen lugar mientras está activo e incorpora los cambios en el índice que se está generando.

20.8. RESUMEN

- Las bases de datos en paralelo han logrado una aceptación comercial significativa en los últimos 15 años.
- En el paralelismo de E/S las relaciones se dividen entre los discos disponibles para que se pueda realizar la recuperación de datos más rápidamente. Tres técnicas de división utilizadas frecuentemente son la división por turno rotatorio, la división por asociación y la división por rangos.
- El sesgo es un problema importante, especialmente con los grados crecientes de paralelismo. Los vectores de división equilibrados, usando histogramas, y la división con procesadores virtuales son algunas técnicas usadas para reducir el sesgo.
- En el paralelismo entre consultas se ejecutan concurrentemente diferentes consultas para aumentar la productividad.
- El paralelismo en las consultas intenta reducir el coste de ejecutar una consulta. Hay dos tipos de paralelismo en las consultas: el paralelismo en operaciones y el paralelismo entre operaciones.
- El paralelismo en operaciones se utiliza para ejecutar operaciones relacionales, como las ordenaciones y las reuniones, en paralelo. El paralelismo en operaciones es natural en las operaciones relacionales, dado que están orientadas a conjuntos.
- Hay dos enfoques básicos en la paralelización de las operaciones binarias como las reuniones.
  — En el paralelismo de particiones las relaciones se dividen en varias partes y las tuplas de r_i sólo se reúnen con las tuplas de s_i. El paralelismo de particiones se puede usar sólo para las reuniones naturales y las equirreuniones.
  — En los fragmentos y réplicas asimétricos una de las relaciones se replica mientras la otra se divide. A diferencia del paralelismo de particiones, los fragmentos y réplicas pueden utilizarse con cualquier condición de reunión. Ambas técnicas de paralelismo pueden utilizarse en combinación con cualquier técnica de reunión.

  Ambas técnicas de paralelización pueden funcionar en conjunción con cualquier técnica de reunión.
- En el paralelismo independiente las diferentes operaciones que no tienen dependencia entre sí se ejecutan en paralelo.
- En el paralelismo encauzado los procesadores envían los resultados de una operación a otra según los van procesando, sin esperar a que concluya toda la operación.
- La optimización de consultas en bases de datos paralelas es significativamente más compleja que la optimización de consultas en bases de datos secuenciales.

TÉRMINOS DE REPASO

- Atributo de división
- Coherencia caché
- Construcción interactiva de índices
- Consulta concreta
- Consulta de rangos
- Consultas de ayuda a la toma de decisiones
- Coste de la evaluación paralela
- Diseño de sistemas paralelos
- División horizontal
- Eliminación de duplicados en paralelo
- Manejo del sesgo
  – Histograma
  – Procesadores virtuales
  – Vector de división por rangos equilibrado
- Modelo del operador de intercambio
- Optimización de consultas
- Ordenación paralela
  – Ordenación con división por rangos
  – Ordenación y mezcla externas paralelas
- Paralelismo en consultas
  – Paralelismo en operaciones
  – Paralelismo entre operaciones
- Paralelismo entre consultas
- Paralelismo de datos
- Paralelismo de E/S
- Paralelismo entre operaciones
  – Paralelismo de encauzamiento
  – Paralelismo independiente
- Planificación
- Proyección paralela
- Reunión paralela
  – Reunión por asociación dividida en paralelo
  – Reuniones con bucles anidados en paralelo
  – Reunión por división
  – Reunión con fragmentos y réplicas
  – Reunión con fragmentos y réplicas asimétricos

- Selección paralela
- Sesgo
  – Sesgo de la división
  – Sesgo de ejecución
  – Sesgo de los valores de los atributos
- Técnicas de división
  – División por asociación
  – División por rangos
  – Turno rotatorio
- Vector de división

EJERCICIOS

20.1. Para cada una de las tres técnicas de división, es decir, por turno rotatorio, por asociación y por rangos, dese un ejemplo de una consulta para la que esa técnica de división proporcione la respuesta más rápida.

20.2. Al llevar a cabo una selección de rango en un atributo dividido por rangos es posible que sólo haga falta tener acceso a un disco. Descríbanse las ventajas e inconvenientes de esta propiedad.

20.3. Indíquense los factores que puedan dar lugar a sesgo cuando se divide una relación basándose en uno de sus atributos utilizando:

a. División por asociación
b. División por rangos

En cada uno de los casos anteriores, indíquese lo que se puede hacer para reducir el sesgo.

20.4. Indíquese la forma de paralelismo (entre consultas, entre operaciones o en operaciones) que sea probablemente la más importante para cada una de las tareas siguientes.

a. Incrementar la productividad de un sistema con muchas consultas pequeñas.
b. Incrementar la productividad de un sistema con unas pocas consultas grandes cuando el número de discos y de procesadores es grande.

20.5. Con el paralelismo de encauzamiento suele resultar adecuado llevar a cabo varias operaciones de un cauce en un mismo procesador, aunque haya disponibles muchos procesadores.

a. Explíquese el motivo.
b. ¿Serían válidos los argumentos anteriores si la máquina utilizara una arquitectura de memoria compartida? Explíquese.
c. ¿Serían válidos los argumentos anteriores con paralelismo independiente? (Es decir, ¿hay casos en que incluso si las operaciones no se encauzan y hay muchos procesadores disponibles sigue siendo conveniente llevar a cabo varias operaciones en el mismo procesador?)

20.6. Dese un ejemplo de una reunión, que no sea una equirreunión simple, para la que pueda utilizarse paralelismo de particiones. ¿Qué atributos deberían utilizarse para la división?

20.7. Considérese el procesamiento de reuniones utilizando fragmentos y réplicas simétricos con división por rangos. ¿Cómo se puede optimizar la evaluación si la condición de reunión es de la forma $|r.A - s.B| \leq k$, donde k es una constante pequeña. Aquí, $|x|$ denota el valor absoluto de x. Una reunión con una condición de reunión así se denomina **reunión de banda**.

20.8. Descríbase una buena manera de paralelizar lo siguiente.

a. La operación diferencia.
b. La agregación utilizando la operación **count**.
c. La agregación utilizando la operación **count distinct**.
d. La agregación utilizando la operación **avg**.
e. La reunión externa por la izquierda, si la condición de reunión sólo implica igualdad.
f. La reunión externa por la izquierda, si la condición de reunión implica comparaciones distintas de la igualdad.
g. La reunión externa completa, si la condición de reunión implica comparaciones distintas de la igualdad.

20.9. Recuérdese que los histogramas se utilizan para generar particiones en rangos con carga equilibrada.

a. Supóngase que se tiene un histograma en el que los valores varían de 1 a 100 y están divididos en 10 rangos, 1-10, 11-20, …, 91-100, con las frecuencias 15, 5, 20, 10, 10, 5, 5, 20, 5 y 5, respectivamente. Dese una función de división por rangos con carga equilibrada para dividir los valores en cinco particiones.
b. Propóngase un algoritmo para procesar una división por rangos con carga equilibrada con p particiones, dado un histograma de las distribuciones de frecuencias que contiene n rangos.

20.10. Descríbanse las ventajas e inconvenientes de utilizar paralelismo de encauzamiento.

20.11. Algunos sistemas paralelos de bases de datos guardan otra copia de cada elemento de los datos en discos conectados a un procesador diferente para evitar la pérdida de los datos si falla uno de los procesadores.

a. ¿Por qué es conveniente dividir las copias de los elementos de los datos de un procesador entre varios procesadores?
b. ¿Cuáles son las ventajas e inconvenientes de utilizar almacenamiento RAID en lugar de guardar otra copia de cada elemento de datos?

NOTAS BIBLIOGRÁFICAS

Los sistemas de bases de datos relacionales comenzaron a aparecer en el mercado en 1983; ahora lo dominan. A finales de los 70 y comienzos de los 80, a medida que el modelo relacional lograba un fundamento razonablemente sólido, la gente reconoció que se puede aprovechar un elevado nivel de paralelismo en los operadores de relación y tienen buenas propiedades de flujo de datos. Se lanzaron en rápida sucesión un sistema comercial, Teradata, y varios proyectos de investigación, como GRACE (Kitsuregawa et al. [1983], Fushimi et al. [1986]), GAMMA (DeWitt et al. [1986, 1990]) y Bubba (Boral et al. [1990]). Los investigadores utilizaron estos sistemas paralelos de bases de datos para estudiar la viabilidad de la ejecución en paralelo de los operadores relacionales. Posteriormente, a finales de los 80 y en los 90, varias compañías más –como Tandem, Oracle, Sybase, Informix y Red-Brick (ahora parte de Informix, que a su vez es parte de IBM)– han entrado en el mercado de bases de datos paralelas. Los proyectos de investigación en el mundo académico incluyen XPRS (Stonebraker et al. [1989]) y Volcano (Graefe [1990]).

El bloqueo en las bases de datos paralelas se discute en Joshi [1991] y en Mohan y Narang [1991, 1992b]. Los protocolos sobre coherencia de caché para los sistemas paralelos de bases de datos se discuten en Dias et al. [1989], Mohan y Narang [1991], Mohan y Narang [1992] y en Rahm [1993]. Carey et al [1991] discute los asuntos de caché en sistemas cliente-servidor. El paralelismo y la recuperación en sistemas de bases de datos se discuten en Bayer et al. [1980].

Graefe [1993] presenta una excelente visión general del procesamiento de consultas, incluyendo el procesamiento de consultas en paralelo. La ordenación en paralelo se discute en DeWitt et al. [1992]. Los algoritmos de reuniones en paralelo se describen en Nakayama et al. [1984], Kitsuregawa et al. [1983] y Richardson et al. [1987], Schneider y DeWitt [1989], Kitsuregawa y Ogawa [1990], Lin et al. [1994] y Wilschut et al. [1995], entre otros trabajos. Los algoritmos para la reunión en paralelo para arquitecturas de memoria compartida se describen en Tsukuda et al. [1992], Deshpande y Larson [1992] y Shatdal y Naughton [1993].

El tratamiento del sesgo en las reuniones en paralelo se describe en Walton et al. [1991], Wolf [1991] y DeWitt et al. [1992]. Las técnicas de muestreo para bases de datos paralelas se describen en Seshadri y Naughton [1992] y Ganguly et al. [1996]. Graefe [1990] y Graefe [1993] defendieron el modelo del operador de intercambio.

Las técnicas de optimización de las consultas en paralelo se describen en H. Lu y Tan [1991], Hong y Stonebraker [1991], Ganguly et al. [1992], Lanzelotte et al. [1993], Hasan y Motwani [1995] y Jhingran et al. [1997].

PARTE

VII

OTROS TEMAS

El Capítulo 21 trata varios aspectos sobre la creación y el mantenimiento de aplicaciones y de la administración de sistemas de bases de datos. El capítulo describe en primer lugar el modo de implementar las interfaces de usuario, en especial las interfaces basadas en Web. Otros aspectos como el ajuste del rendimiento (para mejorar la velocidad de las aplicaciones), la normalización en el comercio electrónico y el modo de tratar los sistemas heredados se tratan también en este capítulo.

El Capítulo 22 describe varios avances recientes en la realización de consultas y la recuperación de la información. En primer lugar trata las extensiones de SQL para dar soporte a nuevos tipos de consultas, en especial a las consultas que suelen formular los analistas de datos. A continuación trata el almacenaje de datos, en el que los datos generados por las diferentes partes de una organización se reúnen de manera centralizada. El capítulo describe luego la recopilación de datos, que pretende hallar estructuras de diversas formas complejas en grandes volúmenes de datos. Finalmente, el capítulo describe la recuperación de la información, que trata de las técnicas para consultar conjuntos de documentos de texto, como las páginas Web, para hallar los documentos de interés.

El Capítulo 22 describe los tipos de datos, como los datos temporales, los espaciales y los multimedia, y los aspectos del almacenamiento de dichos datos en las bases de datos. Las aplicaciones como la informática móvil y sus conexiones con las bases de datos también se describen en este capítulo.

Finalmente, el Capítulo 23 describe varias técnicas avanzadas de procesamiento de transacciones, incluidos los monitores de procesamiento de transacciones, los flujos de trabajo transaccionales, las transacciones de larga duración y las transacciones entre varias bases de datos.

CAPÍTULO

21 DESARROLLO DE APLICACIONES Y ADMINISTRACIÓN

Casi todo el uso de las bases de datos se produce desde los programas de aplicaciones. A su vez, casi toda la interacción con los usuarios con las bases de datos es indirecta, mediante los programas de aplicaciones. No resulta sorprendente, por tanto, que los sistemas de bases de datos lleven mucho tiempo soportando herramientas como los generadores de formularios y de interfaces gráficas de usuario, que ayudan a lograr el desarrollo rápido de aplicaciones que actúan de interfaz con los usuarios. En los últimos años la Web se ha transformado en la interfaz de usuario con las bases de datos más utilizada.

Una vez creada una aplicación suele descubrirse que se ejecuta más lenta de lo que deseaban los diseñadores o que maneja menos transacciones por segundo de lo necesario. Se puede hacer que las aplicaciones se ejecuten significativamente más rápido mediante el ajuste del rendimiento, que consiste en hallar y eliminar los cuellos de botella y en añadir el hardware adecuado, como puede ser memoria o discos. Los índices de calidad ayudan a caracterizar el rendimiento de los sistemas de bases de datos.

Las normas son muy importantes para el desarrollo de las aplicaciones, especialmente en la época de Internet, dado que éstas necesitan comunicarse entre sí para llevar a cabo tareas útiles. Se han propuesto varias normas que afectan al desarrollo de las aplicaciones de bases de datos.

El comercio electrónico se está convirtiendo en parte integral del modo en que se adquieren bienes y servicios y las bases de datos desempeñan un papel importante en ese dominio.

Los sistemas heredados son sistemas basados en tecnología de generaciones anteriores. Suelen hallarse en el núcleo de las organizaciones y ejecutan aplicaciones con misiones críticas. Se describen aspectos del establecimiento de interfaces con los sistemas heredados y el modo en que pueden sustituirse por otros sistemas.

21.1. INTERFACES WEB PARA BASES DE DATOS

La **World Wide Web** (**Web**, para abreviar) es un sistema distribuido de información basado en hipertexto. Las interfaces Web con las bases de datos se han vuelto muy importantes. Tras describir varios motivos para establecer interfaces con la Web (Apartado 21.1.1) se ofrece un resumen de la tecnología Web (Apartado 21.1.2) y se estudian los servidores Web (Apartado 21.1.3) y se describen algunas técnicas de primera línea para la creación de interfaces Web con bases de datos, mediante servlets y lenguajes de guiones del lado del servidor (Apartados 21.1.4 y 21.1.5). En el Apartado 21.1.6 se describen técnicas para mejorar el rendimiento.

21.1.1. Motivación

La Web se ha vuelto importante como frontal para las bases de datos por varios motivos: los navegadores Web ofrecen un frontal *universal* para la información facilitada por los dorsales ubicados en cualquier parte del mundo. El frontal puede ejecutarse en cualquier sistema informático y no hace falta que el usuario descargue software específico para que tenga acceso a la información. Además, hoy en día casi todos los que pueden permitírselo tienen acceso a la Web.

Con el crecimiento de los servicios de información y del comercio electrónico en la Web, las bases de datos empleadas para los servicios de información, el ayuda a la toma de decisiones y el procesamiento de transacciones deben estar conectadas a la Web. La interfaz de los formularios HTML resulta conveniente para el procesamiento de las transacciones. El usuario puede rellenar los detalles de un formulario de pedido y pulsar un botón de envío para remitir un mensaje al servidor, el cual ejecuta un programa de aplicación correspondiente al formulario de pedido y esta acción, a su vez, ejecutará las transacciones en una base de datos en el sitio del servidor. El servidor da formato a los resultados de la transacción y se los devuelve al usuario.

Otro motivo para el uso de interfaces entre las bases de datos y la Web es que la presentación exclusiva de documentos estáticos (fijos) en un sitio Web presenta

algunas limitaciones, aunque el usuario no realice ninguna consulta ni procese ninguna transacción:

- Los documentos Web fijos no permiten que lo que se muestre se adapte al usuario. Por ejemplo, puede que un periódico desee personalizar su aspecto para cada usuario, para destacar los artículos informativos que con mayor probabilidad interesen al usuario.
- Cuando se actualizan los datos de la compañía, los documentos de la Web se quedan obsoletos si no se actualizan de manera simultánea. El problema se hace más grave si varios documentos de la Web replican datos importantes y hay que actualizarlos todos.

Estos problemas pueden resolverse mediante la generación dinámica de los documentos de la Web a partir de una base de datos. Cuando se solicita un documento se ejecuta un programa en el sitio del servidor, que a su vez ejecuta consultas en una base de datos y genera el documento solicitado de acuerdo con los resultados de las consultas. Siempre que se actualizan datos importantes de la base de datos los documentos generados se actualizan de manera automática. El documento generado también puede personalizarse para el usuario de acuerdo con la información del usuario guardada en la base de datos.

Las interfaces Web ofrecen ventajas atrayentes incluso para aplicaciones de bases de datos que sólo se utilizan dentro de una organización. La norma **lenguaje de marcas de hipertexto (HyperText Markup Language, HTML)** permite que el texto reciba formato de manera ordenada y que se destaque la información importante. Los **hipervínculos**, que son enlaces con otros documentos, pueden asociarse con regiones de los datos mostrados. Al pulsar en un hipervínculo se captura el documento vinculado y se muestra. Los hipervínculos resultan muy útiles para explorar datos, lo que permite a los usuarios obtener más detalles de las partes de los datos que prefieran.

Finalmente, los navegadores de hoy en día pueden capturar programas junto con los documentos HTML, y ejecutarlos en el navegador, en modo seguro, es decir, sin dañar los datos de la computadora del usuario. Los programas pueden escribirse en lenguajes de guiones del lado del cliente, como Javascript, o ser *applets* escritas en el lenguaje Java. Estos programas permiten la creación de interfaces de usuario sofisticadas, más de lo que es posible únicamente con HTML, interfaces que pueden utilizarse sin descargar ni instalar ningún software. Por ello, las interfaces Web resultan potentes y atractivas visualmente, y es posible que eclipsen las interfaces específicas salvo para una pequeña clase de usuarios.

21.1.2. Fundamentos de la Web

En este apartado se va a repasar parte de la tecnología básica tras la World Wide Web, para los lectores que no estén familiarizados con ella.

21.1.2.1. Los localizadores uniformes de recursos

Un **localizador uniforme de recursos (Uniform Resource Locator, URL)** es un nombre globalmente único para cada documento al que se puede tener acceso en la Web. Un ejemplo de URL es

http://www.bell-labs.com/topic/books/db-book

La primera parte del URL indica el modo en que se puede tener acceso al documento: *http* indica que se puede tener acceso al documento mediante el protocolo de transferencia de hipertexto (HyperText Transfer Protocol, HTTP), que es un protocolo para transferir documentos HTML. La segunda parte da el nombre único de una máquina que tiene un servidor Web. El resto del URL es el nombre del camino hasta el archivo en la máquina, u otro identificador único del documento dentro de la máquina.

Muchos datos de la Web se generan de manera dinámica. Un URL puede contener el identificador de un programa ubicado en la máquina servidora Web, así como argumentos que hay que darle al programa. Un ejemplo de URL de este tipo es

http://www.google.com/search?q=silberschatz

que dice que el programa search en el servidor www.google.com se debe ejecutar con el argumento q=silberschatz. El programa se ejecuta, utilizando los argumentos dados, y devuelve un documento HTML, que se envía al frontal.

21.1.2.2. El lenguaje de marcas de hipertexto

La Figura 21.1 es un ejemplo de origen de un documento HTML. La Figura 21.2 muestra la imagen que crea este documento.

```
<html>
<body>
<table BORDER COLS=3>
<tr> <td>C-101</td> <td>Centro</td> <td>500</td> </tr>
<tr> <td>C-102</td> <td>Navacerrada</td> <td>400</td> </tr>
<tr> <td>C-201</td> <td>Galapagar </td> <td>900</td> </tr>
</table>
<center> La <i>relación</i> cuenta </center>

<form action="BankQuery" method=get>
Seleccione cuenta/préstamo e introduzca el número <br>
<select name="tipo">
    <option value="cuenta" seleccionada>Cuenta
    <option value="préstamo"> Préstamo
</select>
<input type=text size=5 name="número">
<input type=submit value="Enviar">

</body>
</html>
```

FIGURA 21.1. Texto fuente HTML.

C—101	Centro	500
C—102	Navacerrada	400
C—201	Galapagar	900

La relación *cuenta*



FIGURA 21.2. Visualización del texto fuente HTML de la Figura 21.1.

Las figuras muestran el modo en que HTML puede mostrar una tabla y un formulario sencillo que permite que los usuarios seleccionen el tipo (cuenta o préstamo) de un menú e introducir un número en un cuadro de texto. HTML también soporta otros tipos de entrada. Al pulsar en el botón de envío se hace que el programa BankQuery (especificado en el campo form action) se ejecute con los valores proporcionados por el usuario para los argumentos type y number (especificados en los campos select e input). El programa genera un documento HTML, que se devuelve al usuario y se le muestra; se verá el modo de crear estos programas en los Apartados 21.1.3, 21.1.4 y 21.1.5.

HTML soporta *hojas de estilo*, que pueden modificar las definiciones predeterminadas del modo en que se muestran las estructuras de formato de HTML, así como otros atributos de exhibición como el color de fondo de la página. La norma *hojas de estilo en cascada (cascading stylesheet, css)* permite que la misma hoja de estilo se utilice para varios documentos HTML, lo que da un aspecto uniforme a todas las páginas del sitio Web.

21.1.2.3. Guiones del lado del cliente y applets

La inclusión de código de programa en los documentos permite que las páginas Web sean activas y ejecuten actividades como la animación ejecutando programas en el sitio local, en lugar de presentar simplemente textos pasivos y gráficos. El uso principal de estos programas es la interacción flexible con el usuario, más allá de la limitada capacidad de interacción proporcionada por HTML y por los formularios HTML. Además, la ejecución de programas en el sitio del cliente acelera mucho la interacción, en comparación con tener que enviar cada interacción al sitio del servidor para su procesamiento.

Un riesgo de dar soporte a esos programas es que, si el diseño del sistema no se realiza con cuidado, el código de programa incluido en la página Web (o, de modo equivalente, en un mensaje de correo electrónico) puede realizar acciones malévolas en la computadora del usuario. Las acciones malévolas pueden variar desde la lectura de información privada o la eliminación o la modificación de información de la computadora hasta la toma del control de la computadora y la difusión del código a otras computadoras (por correo electrónico, por ejemplo). En los últimos años varios virus de correo electrónico se han difundido ampliamente de este modo.

El lenguaje *Java* se ha hecho muy popular porque proporciona un modo seguro de ejecutar los programas en las computadoras de los usuarios. El código Java puede compilarse en «código de bytes» independiente de la plataforma que puede ejecutarse en cualquier explorador que soporte Java. A diferencia de los programas locales, los programas de Java (*applets*) descargados como parte de una página Web carecen de autoridad para llevar a cabo acciones que puedan resultar destructivas. Se les permite mostrar datos en la pantalla o establecer una conexión de red para obtener más información con el servidor desde el que se ha descargado la página Web. Sin embargo, no se les permite el acceso a los archivos locales, ejecutar programas del sistema ni establecer conexiones de red con otras computadoras.

Aunque Java es un lenguaje de programación completo, hay lenguajes más sencillos, denominados **lenguajes de guiones**, que pueden enriquecer la interacción con el usuario, mientras ofrecen la misma protección que Java. Estos lenguajes proporcionan estructuras que pueden incluirse en los documentos de HTML. **Los lenguajes de guiones del lado del cliente** son lenguajes diseñados para ejecutarse en el explorador Web del cliente. De entre ellos, el lenguaje *Javascript* es, con mucho, el más utilizado. También existen lenguajes de guiones con finalidades especiales para tareas especializadas como la animación (por ejemplo, Flash y Shockwave de Macromedia) y el modelado tridimensional (lenguaje de marcas de realidad virtual, Virtual Reality Markup Language, VRML). Los lenguajes de guiones también pueden utilizarse en el lado del servidor, como se verá más adelante.

21.1.3. Los servidores Web y las sesiones

Los **servidores Web** son programas que se ejecutan en la máquina servidora, que aceptan las solicitudes de los exploradores Web y devuelven los resultados en forma de documentos HTML. El explorador y el servidor Web se comunican mediante un protocolo denominado **protocolo de transferencia de hipertexto (HyperText Transfer Protocol, HTTP)**. HTTP proporciona características potentes, más allá de la mera transferencia de documentos. La característica más importante es la posibilidad de ejecutar programas con los argumentos proporcionados por el usuario y de devolver los resultados como documentos HTML.

En consecuencia, los servidores Web pueden actuar con facilidad como intermediarios para ofrecer acceso a diversos servicios de información. Se pueden crear servicios nuevos mediante la creación e instalación de programas de aplicaciones que ofrezcan los servicios. La norma **interfaz de pasarela común (common gate-**

way interface, CGI) define el modo en que el servidor Web se comunica con los programas de las aplicaciones. Los programas de las aplicaciones suelen comunicarse con servidores de bases de datos, mediante ODBC, JDBC u otros protocolos, con objeto de obtener o guardar datos.

La Figura 21.3 muestra un servicio Web que emplea una arquitectura de tres capas, con un servidor Web, un servidor de aplicaciones y un servidor de bases de datos. El empleo de varios niveles de servidores incrementa la sobrecarga del sistema; la interfaz CGI inicia un proceso nuevo para atender cada solicitud, lo que supone una sobrecarga todavía mayor.

Por tanto, la mayor parte de los servicios Web de hoy en día utilizan una arquitectura Web de dos capas, en la que el programa de la aplicación se ejecuta en el servidor Web, como en la Figura 21.4. En los apartados siguientes se estudiarán con más detalle los sistemas basados en arquitectura de dos capas.

Hay que tener en cuenta que no existe una conexión continua entre el cliente y el servidor. Por el contrario, cuando un usuario inicia una sesión en una computadora o se conecta con un servidor ODBC o JDBC, se crea una sesión y en el servidor y en el cliente, se conserva la información de la sesión hasta que ésta concluya (información tal como si se autentificó al usuario mediante contraseña o las opciones de sesión definidas por el usuario). El motivo de que HTTP sea **sin conexión** es que la mayor parte de las computadoras presentan límites en el número de conexiones simultáneas que pueden aceptar; y si gran número de sitios de la Web abren conexiones, se podría superar ese límite y denegarse el servicio a otros posteriores. Con un servicio sin conexión, ésta se interrumpe en cuanto se satisface una solicitud, lo que deja las conexiones disponibles para otras solicitudes.

La mayor parte de los servicios de información necesitan información sobre las sesiones. Por ejemplo, los servicios suelen restringir el acceso a la información y, por tanto, necesitan autentificar a los usuarios. La autenticación debe hacerse una vez por sesión, y las interacciones posteriores de la sesión no deben exigir que se repita la autenticación.

Para crear la vista de estas sesiones hay que guardar en el cliente información adicional, y devolverla con cada solicitud de la sesión, para que el servidor identifique que la solicitud forma parte de la sesión del usuario. En el servidor también hay que guardar información adicional sobre la sesión.

Esta información adicional se conserva en el cliente en forma de **cookie**; una cookie no es más que un pequeño fragmento de texto que contiene información de identificación. El servidor envía una cookie al cliente tras la autenticación, y también guarda una copia de modo local. Las cookies enviadas a clientes diferentes contienen texto de identificación diferente. El explorador envía las cookies de manera automática en posteriores solicitudes de documentos al mismo servidor. Mediante la comparación de la cookie con las cookies guardadas de manera local en el servidor, éste puede identificar la solicitud como integrante de una sesión activa. Las cookies también pueden utilizarse para guardar las preferencias de los usuarios y emplearlas cuando el servidor responda a una solicitud. Las cookies pueden guardarse en el explorador de manera permanente; identifican al usuario en visitas posteriores al mismo sitio, sin que haya que escribir ninguna identificación.

21.1.4. Servlets

En la arquitectura Web de dos capas las aplicaciones se ejecutan como parte del propio servidor Web. Un modo de implementar esta arquitectura es cargar los programas Java con el servidor Web. La especificación **servlet** de Java define una interfaz de programación de aplicaciones para la comunicación entre el servidor Web y

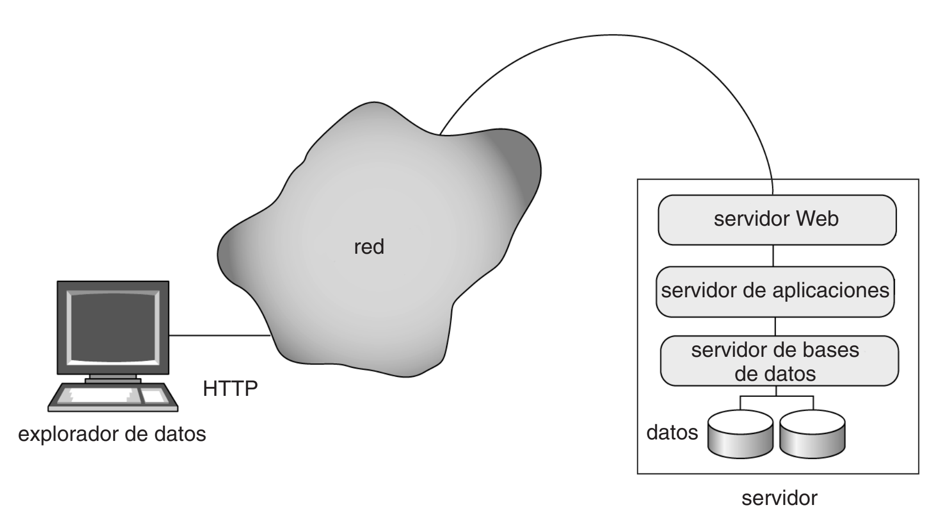


FIGURA 21.3. Arquitectura de tres capas.

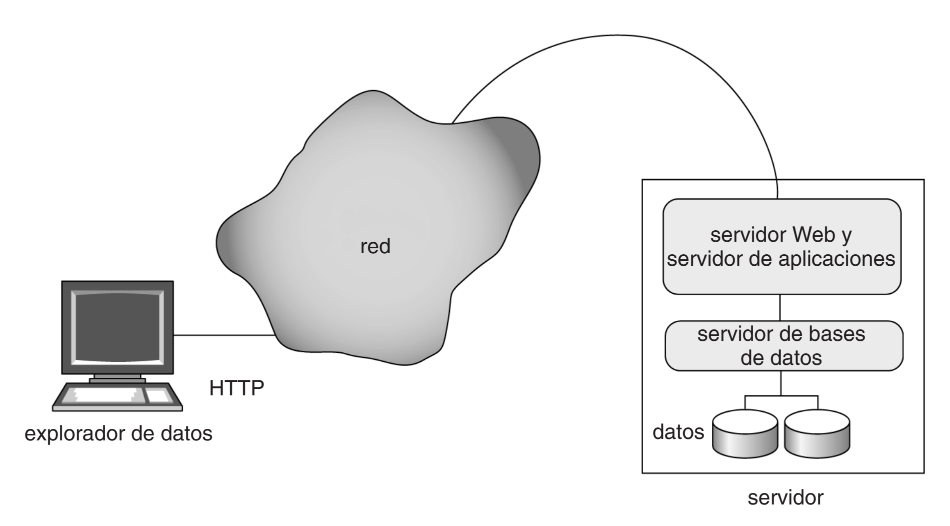


FIGURA 21.4. Arquitectura de dos capas.

los programas de aplicaciones. La palabra **servlet** también hace referencia a un programa Java que implementa la interfaz servlet. El programa se carga en el servidor Web cuando éste se inicia o cuando el servidor recibe una solicitud Web para ejecutar la aplicación servlet. La Figura 21.5 es un ejemplo de código servlet para implementar el formulario de la Figura 21.1.

El servlet se denomina BankQueryServlet, mientras que el formulario especifica que action=«Bank-Query». Se debe indicar al servidor Web que este servlet se va a utilizar para tratar las solicitudes para BankQuery.

El ejemplo dará una idea del modo en que se emplean los servlets. Para obtener los detalles necesarios para crear una aplicación servlet se puede consultar un libro sobre servlets o leer la documentación en línea referente a servlets que forma parte de la documentación de Java que facilita Sun. Véanse las notas bibliográficas para obtener referencias de estas fuentes.

El formulario especifica que se emplea el mecanismo HTTP get para transmitir los parámetros (post es el otro mecanismo más utilizado). Así se invoca el método doGet() del servlet, que se define en el código. Cada solicitud da lugar a una nueva hebra en la que se ejecuta la llamada, por lo que se pueden tratar en paralelo varias solicitudes.

Los valores de los menús del formulario y de los campos de entrada de la página Web, así como las cookies, pasan por un objeto de la clase HttpServletRequest que se crea para la solicitud, y la respuesta a la solicitud pasa por un objeto de la clase HttpServlet-Response[1].

El código del método doGet() del ejemplo extrae los valores del tipo y del número del parámetro mediante request.getParameter(), y los utiliza para realizar una consulta a la base de datos. El código empleado para tener acceso a la base de datos no se muestra; hay que consultar el Apartado 4.13.2 para conseguir detalles del modo de empleo de JDBC para tener acceso a las bases de datos. El sistema devuelve el resultado de la consulta al solicitante imprimiéndolos en formato HTML en Http-Servlet-Response result.

La API del servlet ofrece un método conveniente de creación de sesiones. Al llamar al método getSession(true) de la clase HttpServletRequest se crea un objeto nuevo del tipo HttpSession si se trata de la primera solicitud de ese cliente; el argumento true indica que hay que crear una sesión si se trata de una solicitud nueva. El método devuelve un objeto ya existente si ya se había creado para esa sesión del explorador. Internamente se utilizan cookies para reconocer que una solicitud proviene de la misma sesión del explorador que otra anterior. El código de servlet puede guardarse y buscar parejas (nombre-atributo, valor) en el objeto HttpSession, para conservar el estado para varias solicitudes. Por ejemplo, puede que la primera solicitud de una sesión solicite la identidad y la contraseña del usuario y guarde la identidad del usuario en el objeto sesión. En solicitudes posteriores desde esa sesión del navegador la identidad del usuario se hallará en el objeto sesión.

La visualización de los resultados de las consultas es una labor común de muchas aplicaciones de bases de datos. Se puede crear una función genérica que tome como argumento cualquier conjunto de resultados de JDBC y muestre adecuadamente las tuplas en el conjunto de resultados. Las llamadas a los metadatos de JDBC pueden utilizarse para buscar informaciones en el resultado de la consulta como el número de columnas y el nombre y el tipo de las columnas; esta información se utiliza luego para imprimir el resultado de la consulta.

[1] La interfaz servlet también puede soportar solicitudes que no sean HTTP, aunque los ejemplos de este libro sólo utilicen HTTP.

```
public class BankQueryServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse result)
        throws ServletException, IOException
    {
        String type = request.getParameter("type");
        String number = request.getParameter("number");
        ... código para buscar el saldo del préstamo/cuenta ...
        ... empleo de JDBC para comunicarse con la base de datos ...
        ... se da por supuesto que el valor se guarda en la variable balance ...

        result.setContentType("text/html");
        PrintWriter out = result.getWriter();
        out.println("<HEAD><TITLE> Resultado de la consulta</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("Saldo a " + type + number + " = " + balance);
        out.println("</BODY>");
        out.close();
    }
}
```

FIGURA 21.5. Ejemplo de código servlet.

21.1.5. Guiones del lado del servidor

La escritura incluso de una mera aplicación Web en un lenguaje de programación como Java o C supone una tarea que consume bastante tiempo y necesita muchas líneas de código y programadores familiarizados con las complejidades del lenguaje. Un enfoque alternativo, el de los **guiones del lado del servidor**, ofrece un método mucho más sencillo de creación de muchas aplicaciones. Los lenguajes de guiones ofrecen estructuras que pueden incluirse en los documentos HTML. En los guiones del lado del servidor, antes de entregar una página Web, el servidor ejecuta las secuencias incluidas en el contenido HTML de la página. Cada secuencia, al ejecutarse, puede generar texto que se añade a la página (o que incluso puede eliminar contenido de la página). El código fuente de los guiones se elimina de la página, de modo que puede que el cliente ni siquiera se dé cuenta de que la página contenía originalmente código. Puede que la secuencia de comandos ejecutada contenga código SQL que se ejecute en una base de datos.

En los últimos años han aparecido varios lenguajes de guiones. Entre ellos están Server-Side Javascript de Netscape, JScript de Microsoft, JavaServer Pages (JSP) de Sun, el preprocesador de hipertexto PHP (PHP Hypertext Preprocessor, PHP), el lenguaje de marcas ColdFusion de ColdFusion (ColdFusion's ColdFusion Markup Language, CFML) y DTML de Zope. De hecho, incluso es posible incluir en las páginas HTML código escrito en lenguajes de guiones más antiguos como VBScript, Perl y Python. Por ejemplo, Active Server Pages (ASP) de Microsoft soporta VBScript y JScript incorporados. Otros enfoques han extendido el software para la escritura de informes, desarrollado originalmente para la generación de informes imprimibles, para generar informes HTML. También soporta los formularios HTML para la obtención de los valores de los parámetros que se utilizan en las consultas incrustadas en los informes.

Claramente, hay muchas opciones entre las que escoger. Todas soportan características similares, pero se diferencian en el estilo de programación y en la facilidad con que se pueden crear aplicaciones sencillas.

21.1.6. Mejora del rendimiento

Miles de millones de personas de todo el globo pueden tener acceso a los sitios Web, con tasas de millares de solicitudes por segundo, o incluso mayores, en el caso de los sitios más populares. Asegurarse de que las solicitudes se atiendan con tiempos de respuesta bajos supone un importante desafío para los desarrolladores de los sitios Web.

Se emplean técnicas de almacenamiento caché de varios tipos para aprovechar las características comunes de las diversas transacciones. Por ejemplo, supóngase que el código de la aplicación para la atención de cada solicitud necesita contactar con una base de datos mediante JDBC. La creación de una nueva conexión JDBC puede tardar varios milisegundos, por lo que la apertura de una conexión nueva para cada solicitud no es una buena idea si hay que soportar tasas de transacción muy elevadas. Muchas aplicaciones crean un fondo de conexiones JDBC abiertas y cada solicitud utiliza una de las conexiones de ese fondo.

Puede que muchas solicitudes den por resultado exactamente la misma consulta que se ejecutará en la base de datos. El coste de la comunicación con la base de datos puede reducirse enormemente guardando en la caché los resultados de consultas anteriores y volviendo a utilizarlas, en tanto en cuanto no haya cambiado en la base de datos el resultado de la consulta. Algunos servidores Web soportan este almacenamiento en la caché de los resultados de las consultas.

Se pueden reducir aún más los costes guardando en la caché la página Web final que se envía en respuesta a una consulta. Si llega una consulta nueva exactamente con los mismos parámetros que una consulta anterior

y la página Web resultante se halla en la caché, puede volver a utilizarse, lo que evita el coste de volver a calcular la página.

Los resultados de las consultas y las páginas Web guardados en la caché son modalidades de vistas materializadas. Si la base de datos subyacente se modifica, pueden descartarse o volver a calcularse o, incluso, actualizarse de modo incremental, como en el mantenimiento de las vistas materializadas (Apartado 14.5). Por ejemplo, el servidor Web de IBM que se empleó en los Juegos Olímpicos de 2000 puede realizar el seguimiento de los datos de los que depende una página Web guardada en la caché y volver a calcular la página si se modifican esos datos.

21.2. AJUSTE DEL RENDIMIENTO

El ajuste del rendimiento de un sistema implica el ajuste de varios parámetros y opciones de diseño para mejorar su rendimiento para una aplicación concreta. Varios aspectos del diseño de los sistemas de bases de datos (que van desde aspectos de alto nivel como el esquema y el diseño de las transacciones hasta parámetros de las bases de datos como los tamaños de la memoria intermedia o aspectos del hardware como el número de discos) afectan al rendimiento de las aplicaciones. Cada uno de estos aspectos puede ajustarse de modo que se mejore el rendimiento.

21.2.1. Localización de los cuellos de botella

El rendimiento de la mayor parte de los sistemas (al menos, antes de ajustarlos) suele quedar limitado principalmente por el que presenta un componente o unos pocos, denominados **cuellos de botella**. Por ejemplo, puede que un programa pase el 80 por ciento del tiempo en un pequeño bucle ubicado en las profundidades del código y el 20 por ciento restante del tiempo en el resto del código; ese pequeño bucle es, pues, un cuello de botella. La mejora del rendimiento de un componente que no sea un cuello de botella hace poco para mejorar la velocidad global del sistema; en este ejemplo, la mejora de la velocidad del resto del código no puede llevar a más de un 20 por ciento de mejora global, mientras que la mejora de la velocidad del bucle cuello de botella puede lograr una mejora de casi el 80 por ciento global, en el mejor de los casos.

Por tanto, al ajustar un sistema, primero hay que intentar descubrir los cuellos de botella y luego eliminarlos mejorando el rendimiento de los componentes que los generan. Cuando se elimina un cuello de botella puede ocurrir que otro componente se transforme en cuello de botella. En los sistemas bien equilibrados ningún componente aislado constituye un cuello de botella. Si el sistema contiene cuellos de botella se infrautilizan los componentes que no forman parte de los cuellos de botella y quizás pudieran haberse sustituido por componentes más económicos de menores prestaciones.

En los programas sencillos el tiempo pasado en cada zona del código determina el tiempo global de ejecución. No obstante, los sistemas de bases de datos son mucho más complejos y pueden modelarse como **sistemas de colas**. Cada transacción necesita varios servicios del sistema de bases de datos, comenzando por la entrada en los procesos del servidor, las lecturas de disco durante la ejecución, los ciclos de la CPU y los bloqueos para el control de la concurrencia. Cada uno de estos servicios tiene asociada una cola, y puede que las transacciones pequeñas pasen la mayor parte del tiempo esperando en las colas —especialmente en las colas de E/S de los discos— en lugar de ejecutando código. La Figura 21.6 muestra algunas de las colas de los sistemas de bases de datos.

Como consecuencia de las numerosas colas de la base de datos, los cuellos de botella de los sistemas de bases de datos suelen manifestarse en forma de largas colas para un servicio determinado o, de modo equivalente, en elevados índices de utilización de un servicio concreto. Si las solicitudes se espacian de manera exactamente uniforme, y el tiempo para atender una solicitud es menor o igual que el tiempo antes de que llegue la siguiente solicitud, cada solicitud hallará el recurso sin utilizar y podrá iniciar la ejecución de manera inmediata, sin esperar. Por desgracia, la llegada de las solicitudes en los sistemas de bases de datos nunca es tan uniforme, más bien aleatoria.

Si un recurso, como puede ser un disco, tiene un índice de utilización bajo, cuando se realiza una solicitud es probable que ese recurso esté sin utilizar, en cuyo caso el tiempo de espera de la solicitud será cero. Suponiendo llegadas distribuidas de forma aleatoria uniforme, la longitud de la cola (y, en consecuencia, el tiempo de espera) aumentará de manera exponencial con la utilización; a medida que la utilización se aproxime al ciento por ciento, la longitud de la cola aumentará abruptamente, lo que dará lugar a tiempos de espera excesivamente elevados. La utilización de los recursos debe mantenerse lo suficientemente baja como para que la longitud de la cola sea corta. Como indicativo, las utilizaciones cercanas al 70 por ciento se consideran buenas, y las utilizaciones superiores al 90 por ciento se consideran excesivas, dado que generan retrasos significativos. Para aprender más sobre la teoría de los sistemas de colas, generalmente conocida como **teoría de colas**, se pueden consultar las referencias citadas en las notas bibliográficas.

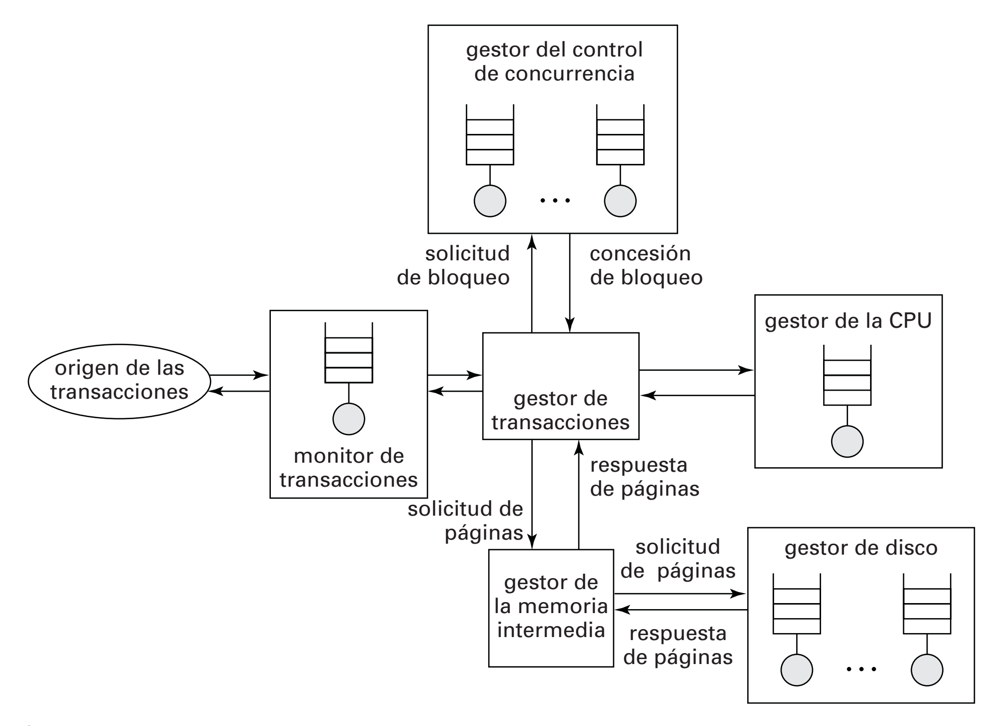


FIGURA 21.6. Colas de un sistema de bases de datos.

21.2.2. Parámetros ajustables

Los administradores de bases de datos pueden ajustar los sistemas de bases de datos en tres niveles. El nivel inferior es el nivel de hardware. Las opciones para el ajuste de los sistemas en este nivel incluyen añadir discos o usar sistemas RAID (si la E/S de disco constituye un cuello de botella), añadir más memoria si el tamaño de la memoria intermedia de disco constituye un cuello de botella o aumentar la velocidad del procesador si el empleo de la CPU constituye un cuello de botella.

El segundo nivel consiste en los parámetros de los sistemas de bases de datos, como el tamaño de la memoria intermedia y los intervalos de puntos de revisión. El conjunto exacto de los parámetros de los sistemas de bases de datos que pueden ajustarse depende de cada sistema concreto de bases de datos. La mayor parte de los manuales de los sistemas de bases de datos proporcionan información sobre los parámetros del sistema de bases de datos que pueden ajustarse y sobre el modo en que deben escogerse los valores de esos parámetros. Los sistemas de bases de datos bien diseñados llevan a cabo automáticamente todos los ajustes posibles, lo que libera al usuario o al administrador de la base de datos de esa carga. Por ejemplo, en muchos sistemas de bases de datos el tamaño de la memoria intermedia es fijo pero ajustable. Si el sistema ajusta de manera automática el tamaño de la memoria intermedia observando los indicadores como las tasas de fallo de las páginas, el usuario no tendrá que preocuparse por el ajuste del tamaño de la memoria intermedia.

El tercer nivel es el nivel superior. Incluye el esquema y las transacciones. El administrador puede ajustar el diseño del esquema, los índices que se crean y las transacciones que se ejecutan para mejorar el rendimiento. El ajuste en este nivel es, comparativamente, independiente del sistema.

Los tres niveles de ajuste interactúan entre sí; hay que considerarlos en conjunto al ajustar los sistemas. Por ejemplo, el ajuste en un nivel superior puede hacer que el cuello de botella pase del sistema de discos a la CPU o viceversa.

21.2.3. Ajuste del hardware

Incluso en un sistema de procesamiento de transacciones bien diseñado, cada transacción suele tener que realizar al menos unas cuantas operaciones de E/S, si los datos necesarios para la transacción se hallan en el disco. Un factor importante en el ajuste de un sistema de procesamiento de transacciones, es asegurarse de que el subsistema de disco puede admitir la velocidad en que se solicitan las operaciones de E/S. Por ejemplo, los discos de hoy en día tienen un tiempo de acceso de unos diez milisegundos y velocidades de transferencia de veinte megabytes por segundo, lo que da cerca de cien operaciones de E/S de acceso aleatorio de un kilobyte cada una. Si cada transacción necesita exactamente dos operaciones de E/S, cada disco soportará como mucho cincuenta transacciones por segundo. La única manera de soportar más transacciones por segundo es aumentar el número de discos. Si el sistema tiene que soportar n transacciones por segundo y cada una

de ella lleva a cabo dos operaciones de E/S, hay que dividir (o fragmentar de otra manera) los datos entre $n/50$ discos (ignorando el sesgo).

Hay que tener en cuenta que el factor limitador no es la capacidad del disco, sino la velocidad en que se puede tener acceso a los datos aleatorios (limitada, a su vez, por la velocidad a la que se puede desplazar el brazo del disco). El número de operaciones de E/S por transacción puede reducirse almacenando más datos en la memoria. Si todos los datos se hallan en la memoria no habrá operaciones de E/S en el disco salvo por las operaciones de escritura. Guardar los datos utilizados con frecuencia en la memoria reduce el número de operaciones de E/S y compensa el coste extra de la memoria. Guardar los datos utilizados con muy poca frecuencia en la memoria sería un despilfarro, dado que la memoria es mucho más cara que los discos.

El problema es, para una cantidad dada de dinero disponible para gastarlo en discos o en memoria, hallar la mejor manera de gastar el dinero para obtener el número máximo de transacciones por segundo. Una reducción de una operación de E/S por segundo ahorra (precio por unidad de disco) / (accesos por segundo por disco). Por tanto, si se tiene acceso a una página concreta *n* veces por segundo, el ahorro debido a guardarla en la memoria es *n* veces el valor calculado anteriormente. Guardar una página en la memoria cuesta (precio por MB de memoria) / (páginas por MB de memoria). Por tanto, el punto de equilibrio es

$$n * \frac{\textit{precio por unidad de disco}}{\textit{accesos por segundo por disco}} = \frac{\textit{precio por MB de memoria}}{\textit{páginas por MB de memoria}}$$

Se puede reordenar la ecuación y sustituir los valores actuales por cada uno de los parámetros citados más arriba para obtener un valor de *n*; si se tiene acceso a una página con una frecuencia mayor que ésta merece la pena comprar suficiente memoria como para almacenarla. La tecnología de discos y los precios actuales de los discos dan un valor de *n* de alrededor de 1/300 de veces por segundo (o, de manera equivalente, una vez cada cinco minutos) para las páginas a las que se tiene acceso de manera aleatoria.

Este razonamiento se refleja en la recomendación denominada **regla de los cinco minutos**: si una página se usa más de una vez cada cinco minutos se debe guardar en la caché de memoria. En otras palabras, merece la pena comprar suficiente memoria para guardar en la caché todas las páginas a las que se tiene acceso al menos una vez cada cinco minutos de promedio. Para los datos a los que se tiene acceso con menos frecuencia hay que comprar discos suficientes para soportar la tasa de E/S exigida por los datos.

La fórmula para hallar el punto de equilibrio depende de factores, como los costes de los discos y de la memoria, que han cambiado en factores de cien o de mil en la última década. No obstante, resulta interesante observar que los índices de los cambios han sido tales que el punto de equilibrio ha permanecido en unos cinco minutos; ¡la regla de los cinco minutos no se ha vuelto la regla de la hora o la regla del segundo!

Para los datos con acceso secuencial se puede leer un número de páginas por segundo significativamente mayor. Suponiendo que se lee cada vez un megabyte de datos, se obtiene la **regla del minuto**, que indica que los datos con acceso secuencial deben guardarse en la caché de memoria si se utilizan al menos una vez por minuto.

Las recomendaciones sólo tienen en cuenta el número de operaciones de E/S y no tienen en consideración factores como el tiempo de respuesta. Algunas aplicaciones necesitan guardar en la memoria incluso los datos que se utilizan con poca frecuencia para soportar tiempos de respuesta inferiores o similares al tiempo de acceso al disco.

Otro aspecto del ajuste es si se debe utilizar RAID 1 o RAID 5. La respuesta depende de la frecuencia con que se actualicen los datos, dado que RAID 5 es mucho más lento que RAID 1 en las operaciones aleatorias de escritura: RAID 5 necesita dos operaciones de lectura y dos operaciones de escritura para ejecutar una sola solicitud aleatoria de escritura. Si una aplicación realiza *l* operaciones aleatorias de lectura y *e* operaciones aleatorias de escritura por segundo para soportar un intercambio concreto, una implementación de RAID 5 necesitaría $l + 4e$ operaciones de E/S por segundo, mientras que una implementación de RAID 1 necesitaría $l + e$ operaciones de E/S por segundo. Se puede calcular el número de discos necesario para soportar las operaciones de E/S necesarias por segundo dividiendo el resultado del cálculo por cien operaciones de E/S por segundo (para los discos de la generación actual). Para muchas aplicaciones, *l* y *e* son lo bastante grandes como para que $(l + e)/100$ discos puedan guardar con facilidad dos copias de todos los datos. Para esas aplicaciones, si se utiliza RAID 1, ¡el número necesario de discos es realmente menor que el número necesario de discos si se emplea RAID 5! Por tanto, RAID 5 sólo resulta útil cuando los requisitos de almacenamiento de datos son muy grandes, pero las velocidades de E/S y los requisitos de transferencia de datos son pequeños, es decir, para datos muy grandes y muy «fríos».

21.2.4. Ajuste del esquema

Dentro de las restricciones de la forma normal escogida es posible dividir las relaciones verticalmente. Por ejemplo, considérese la relación *cuenta*, con el esquema

cuenta (*número-cuenta, nombre-sucursal, saldo*)

para la que *número-cuenta* es una clave. Dentro de las restricciones de las formas normales (formas normales BCNF y tercera) se puede dividir la relación *cuenta* en dos relaciones:

sucursal-cuenta (*número-cuenta, nombre-sucursal*)
saldo-cuenta (*número-cuenta, saldo*)

Las dos representaciones son lógicamente equivalentes, dado que *número-cuenta* es una clave, pero tienen características de rendimiento diferentes.

Si la mayor parte de los accesos a la información de la cuenta sólo examinan *número-cuenta* y *saldo*, pueden ejecutarse sobre la relación *saldo-cuenta*, y es probable que el acceso resulte algo más rápido, dado que no se captura el atributo *nombre-sucursal*. Por el mismo motivo, cabrán en la memoria intermedia más tuplas de *saldo-cuenta* que las correspondientes tuplas de *cuenta*, lo que vuelve a generar un mayor rendimiento. Este efecto sería especialmente destacado si el atributo *nombre-sucursal* fuera de gran tamaño. Por tanto, un esquema que consistiera en *sucursal-cuenta* y *cuenta-saldo* sería preferible en este caso a otro que consistiera en la relación *cuenta*.

Por otro lado, si la mayor parte de los accesos a la información de la cuenta necesitan tanto *saldo* como *nombre-sucursal*, el empleo de la relación *cuenta* será preferible, dado que se evitará el coste de la fusión de *saldo-cuenta* y *sucursal-cuenta*. Además, la sobrecarga de almacenamiento sería menor, dado que sólo habría una relación y no se replicaría el atributo *número-cuenta*.

Otro truco para mejorar el rendimiento es guardar una **relación desnormalizada**, como puede ser una fusión de *cuenta* y de *impositor*, donde la información sobre los nombres de las sucursales y sobre los saldos se repitiera para cada titular de una cuenta. Hay que realizar más esfuerzo para asegurarse de que la relación es consistente siempre que se realice una actualización. No obstante, una consulta que capture los nombres de los clientes y sus saldos asociados se aceleraría, dado que la fusión de *cuenta* con *impositor* se habría calculado previamente. Si se ejecuta con frecuencia una consulta de este tipo, y hay que llevarla a cabo con la máxima eficiencia posible, la relación desnormalizada puede resultar beneficiosa.

Las vistas materializadas pueden proporcionar las ventajas que ofrecen las relaciones desnormalizadas, al coste de algún almacenamiento extra; el ajuste del rendimiento de las vistas materializadas se describe en el Apartado 21.2.6. Una de las principales ventajas de las vistas materializadas respecto de las relaciones desnormalizadas es que el mantenimiento de la consistencia de los datos redundantes pasa a ser labor del sistema de bases de datos, no del programador. Por tanto, las vistas materializadas resultan preferibles, siempre que las soporte el sistema de bases de datos.

Otro enfoque de la aceleración del cálculo de la fusión sin materializarla es agrupar los registros que coincidirán en la fusión en la misma página del disco. Estas organizaciones agrupadas de archivos se vieron en el Apartado 11.7.2.

21.2.5. Ajuste de los índices

Se pueden ajustar los índices de un sistema para mejorar el rendimiento. Si las consultas constituyen el cuello de botella se las suele poder acelerar creando los índices adecuados en las relaciones. Si lo constituyen las actualizaciones, puede que haya demasiados índices, que hay que actualizar cuando se actualizan las relaciones. La eliminación de índices puede que acelere algunas actualizaciones. La elección del tipo de índice también es importante. Algunos sistemas de bases de datos soportan diferentes tipos de índices, como los índices asociativos y los índices de árboles B. Si las consultas más frecuentes son de rango, es preferible utilizar índices de árboles B a los índices asociativos. Otro parámetro ajustable es la posibilidad de hacer que un índice tenga agrupación. Sólo se puede hacer un índice con agrupación por relación, guardando la relación ordenada por los atributos del índice. Generalmente conviene hacer el índice con agrupación que beneficie al mayor número de consultas y de actualizaciones.

Para ayudar a identificar los índices que se deben crear y el índice (si es que hay alguno) de cada relación que se debe agrupar, algunos sistemas de bases de datos proporcionan *asistentes para el ajuste*. Estas herramientas utilizan el historial de consultas y de actualizaciones (denominado *carga de trabajo*) para estimar los efectos de varios índices en el tiempo de ejecución de las consultas y de las actualizaciones en la carga de trabajo. Las recomendaciones sobre los índices que se deben crear se basan en estas estimaciones.

21.2.6. Uso de las vistas materializadas

El uso de las vistas materializadas puede acelerar enormemente ciertos tipos de consultas, en especial las consultas de agregación. Recuérdese el ejemplo del Apartado 14.5 en que el importe total de los créditos de cada sucursal (obtenido sumando los importes de los créditos de todos los créditos de la sucursal) se solicita con frecuencia. Como se vio en ese apartado, la creación de una vista materializada que guarde el importe total de los créditos de cada sucursal puede acelerar enormemente estas consultas.

Las vistas materializadas deben emplearse con cuidado, no obstante, dado que no sólo supone una sobrecarga de espacio almacenarlas sino que, lo que es más importante, su mantenimiento también supone una sobrecarga de tiempo. En el caso del **mantenimiento inmediato de las vistas**, si las actualizaciones de una transacción afectan a la vista materializada, hay que actualizarla como parte de la misma transacción. Por tanto, puede que la transacción se ejecute más lentamente. En el caso del **mantenimiento diferido de las vistas**, la vista materializada se actualiza posteriormente; hasta que se actualice puede que la vista materializada sea inconsistente con las relaciones de la base de datos. Por ejemplo, puede que la vista materializada se actua-

lice cuando la utilice una consulta o que se actualice de manera periódica. El empleo del mantenimiento diferido reduce la carga de las transacciones de actualización.

Un problema importante es el modo en que se seleccionan las vistas materializadas que hay que mantener. El administrador del sistema puede realizar la selección de modo manual examinando los tipos de consultas de la carga de trabajo y averiguando las consultas que necesitan ejecutarse más rápidamente y las actualizaciones o consultas que pueden ejecutarse más lentamente. A partir del examen, el administrador del sistema puede escoger un conjunto adecuado de vistas materializadas. Por ejemplo, puede que el administrador descubra que se utiliza con frecuencia un agregado determinado y decida materializarlo, o puede que halle que una fusión concreta se calcula con frecuencia y decida materializarla.

Sin embargo, la selección manual resulta tediosa incluso para conjuntos de tipos de consultas moderadamente grandes y puede que resulte difícil realizar una buena selección, dado que exige comprender los costes de diferentes alternativas; sólo el optimizador de consultas puede estimar los costes con una precisión razonable, sin ejecutar realmente la consulta. Por tanto, puede que sólo se halle un buen conjunto de vistas mediante el procedimiento de prueba y error, es decir, materializando una o varias vistas, ejecutando la carga de trabajo y midiendo el tiempo empleado para ejecutar las consultas de la carga de trabajo. El administrador repetirá el proceso hasta que se halle un conjunto de vistas que dé un rendimiento aceptable.

Una opción mejor es proporcionar soporte para la selección de las vistas materializadas desde el interior del propio sistema de bases de datos, integrado con el optimizador de consultas. Algunos sistemas de bases de datos, como Microsoft SQL Server 7.5 y RedBrick Data Warehouse de Informix, ofrecen herramientas para ayudar al administrador de la base de datos en la selección de los índices y de las vistas materializadas. Estas herramientas examinan la carga de trabajo (el historial de consultas y de actualizaciones) y sugiere los índices y las vistas que hay que materializar. El usuario tiene la posibilidad de especificar la importancia de la aceleración de las diferentes consultas, lo que el administrador tiene en cuenta al seleccionar las vistas que vaya a materializar.

La herramienta de selección de vistas materializadas de Microsoft también permite que el usuario formule preguntas del tipo «¿qué pasaría si...?», por lo que el usuario puede escoger una vista y el optimizador estimará el efecto de materializarla en el coste total de la carga de trabajo y en los costes individuales de los diferentes tipos de consultas o de actualizaciones en la carga de trabajo.

De hecho, incluso las técnicas de selección automatizadas se implementan de manera parecida internamente: las diferentes opciones se prueban y para cada una el optimizador de consultas estima los costes y las ventajas de materializarla.

La heurística impaciente de la selección de vistas materializadas opera aproximadamente de este modo: se estiman las ventajas de la materialización de las diferentes vistas y se escoge la vista que ofrece la máxima ventaja o la máxima ventaja por unidad de espacio (es decir, la ventaja dividida por el espacio necesario para guardar la vista). Una vez la heurística ha seleccionado una vista puede que hayan cambiado las ventajas de otras vistas, por lo que la heurística vuelve a calcularlas y escoge la segunda mejor vista para su materialización. El proceso continúa hasta que se agota el espacio de disco disponible para guardar las vistas materializadas o bien el coste del mantenimiento de las vistas supera los límites aceptables.

21.2.7. Ajuste de las transacciones

En este apartado se estudian dos enfoques de la mejora del rendimiento de las transacciones:

- Mejora de la orientación del conjunto
- Reducción de la contención de los bloqueos

En el pasado los optimizadores de muchos sistemas de bases de datos no eran especialmente buenos, por lo que el modo en que se había escrito la consulta tenía gran influencia en el modo en que se ejecutaba y, por tanto, en el rendimiento. Los optimizadores avanzados de hoy en día pueden transformar incluso las consultas mal escritas y ejecutarlas de manera eficiente, por lo que la necesidad de ajustar las consultas una a una es menos importante que en el pasado. No obstante, las consultas complejas que contienen subconsultas anidadas no las suelen optimizar muy bien. La mayor parte de los sistemas proporcionan un mecanismo para averiguar el plan de ejecución preciso de las consultas; esta información puede utilizarse para volver a escribir la consulta de forma que el optimizador pueda trabajar mejor con ella.

En SQL incorporado, si se ejecuta con frecuencia una consulta con valores diferentes de un parámetro puede que resulte de ayuda combinar las llamadas en una consulta más orientada al conjunto que sólo se ejecute una vez. Los costes de comunicación de las consultas SQL pueden resultar elevados en los sistemas cliente-servidor, por lo que la combinación de las llamadas de SQL incorporado resulta de especial ayuda en esos sistemas

Por ejemplo, considérese un programa que pase por cada departamento especificado en una lista invocando una consulta de SQL para buscar los gastos totales del departamento empleando la estructura **group by** en la relación *gastos*(*fecha*, *empleado*, *departamento*, *importe*). Si la relación *gastos* no tiene un índice con agrupación en *departamento*, cada consulta de ese tipo dará lugar a una exploración de la relación. En lugar de eso, se puede utilizar una sola consulta SQL para averiguar los gastos totales de todos los departamentos; la con-

sulta puede evaluarse con una sola exploración. Los departamentos importantes pueden examinarse luego en esta relación temporal (mucho más pequeña) que contiene el agregado. Aunque hay un índice que permita el acceso eficiente a las tuplas de un departamento dado, el empleo de varias consultas SQL puede suponer una gran sobrecarga de comunicaciones en los sistemas cliente-servidor. El coste de comunicaciones puede reducirse empleando una sola consulta SQL, capturando los resultados para el lado cliente y pasando por los resultados para hallar las tuplas necesarias.

Otra técnica muy utilizada en los sistemas cliente-servidor para reducir el coste de las comunicaciones y la compilación de SQL es utilizar procedimientos almacenados, en los que las consultas se guardan en el servidor en forma de procedimientos almacenados, que pueden estar compilados con antelación. Los clientes pueden invocar estos procedimientos almacenados en lugar de comunicar consultas enteras.

La ejecución concurrente de diferentes tipos de transacciones puede llevar a veces a un rendimiento bajo por la contención de los bloqueos. Considérese, por ejemplo, una base de datos bancaria. De día se ejecutan de manera casi continua numerosas transacciones de actualización de pequeño tamaño. Supóngase que se ejecuta al mismo tiempo una consulta de gran tamaño que calcula estadísticas de las sucursales. Si la consulta ejecuta una exploración de la relación puede que bloquee todas las actualizaciones de la relación mientras se ejecuta, y eso puede tener un efecto desastroso en el rendimiento del sistema.

Algunos sistemas de bases de datos (Oracle, por ejemplo) permiten el control de la concurrencia multiversión, por lo que las consultas se ejecutan sobre una instantánea de los datos, y las actualizaciones pueden seguir de manera concurrente. Esta característica debe utilizarse si está disponible. Si no se encuentra disponible, una opción alternativa es ejecutar las consultas de gran tamaño en momentos en que las actualizaciones sean pocas o no haya ninguna. Para las bases de datos que soporten los sitios Web puede que no exista ese periodo de calma para las actualizaciones.

Otra opción es utilizar niveles de consistencia más débiles, por lo que la evaluación de la consulta tendrá un impacto mínimo en las actualizaciones concurrentes, pero no se garantiza que los resultados de la consulta sean consistentes. La semántica de la aplicación determina si son aceptables las respuestas aproximadas (inconsistentes).

Las transacciones de actualización de larga duración pueden crear problemas de rendimiento con los registros del sistema e incrementar el tiempo que éste tarda en recuperarse de las caídas. Si una transacción lleva a cabo muchas actualizaciones puede que el registro del sistema se llene antes incluso de que se complete la transacción, en cuyo caso habrá que retroceder la transacción. Si una transacción de actualización se ejecuta durante mucho tiempo (aunque tenga pocas actualizaciones) puede que bloquee la eliminación de las partes más antiguas del registro, si el sistema de registros no está bien diseñado. Nuevamente, este bloqueo puede llevar a que se llene el registro histórico.

Para evitar estos problemas muchos sistemas de bases de datos imponen límites estrictos para el número de actualizaciones que puede llevar a cabo una sola transacción. Aunque el sistema no imponga estos límites suele resultar de ayuda fraccionar las transacciones de actualización de gran tamaño en conjuntos de transacciones de actualización de menor tamaño siempre que sea posible. Por ejemplo, una transacción que dé un aumento a cada empleado de una gran empresa puede dividirse en una serie transacciones de pequeño tamaño, cada una de las cuales se asigna a un pequeño rango de identidades de empleados. Estas transacciones se denominan **transacciones procesadas por minilotes**. No obstante, las transacciones procesadas por minilotes deben emplearse con cuidado. En primer lugar, si hay actualizaciones concurrentes en el conjunto de empleados puede que el resultado del conjunto de transacciones de menor tamaño no sea equivalente al de la transacción única de gran tamaño. En segundo lugar, si se produce un fallo las transacciones confirmadas habrá aumentado el salario de algunos empleados pero no el de los demás. Para evitar este problema, en cuanto el sistema se recupere del fallo, hay que ejecutar las restantes transacciones del lote.

21.2.8. Simulación del rendimiento

Para comprobar el rendimiento de un sistema de bases de datos incluso antes de instalarlo se puede crear un modelo de simulación del rendimiento de ese sistema. En la simulación se modela cada servicio que aparece en la Figura 21.6, como la CPU, cada disco, la memoria intermedia y el control de concurrencia. En lugar de modelar los detalles de un servicio, puede que el modelo de simulación sólo capture algunos aspectos de cada uno, como el **tiempo de servicio**, es decir, el tiempo que tarda en acabar de procesar una solicitud una vez comenzado el procesamiento. Por tanto, la simulación puede modelar el acceso a disco partiendo sólo del tiempo medio de acceso a disco.

Dado que las solicitudes de un servicio suelen tener que aguardar su turno, cada servicio tiene asociada una cola en el modelo de simulación. Cada transacción consiste en una serie de solicitudes. Las solicitudes se encolan según llegan y se atienden de acuerdo con la política de cada servicio, como puede ser que el primero en llegar sea el primero en ser atendido. Los modelos de los servicios como la CPU y los discos operan conceptualmente en paralelo, para tener en cuenta el hecho de que estos subsistemas operan en paralelo en los sistemas reales.

Una vez creado el modelo de simulación para el procesamiento de las transacciones, el administrador del sistema puede ejecutar en él varios experimentos. El

administrador puede utilizar los experimentos con transacciones simuladas que lleguen con diferentes velocidades para averiguar el modo en que se comportaría el sistema bajo diferentes condiciones de carga. También puede ejecutar otros experimentos que varíen los tiempos de servicio de cada servicio para averiguar la sensibilidad del rendimiento a cada uno de ellos. También se pueden variar los parámetros del sistema de modo que se pueda realizar el ajuste del rendimiento en el modelo de simulación.

21.3. PRUEBAS DE RENDIMIENTO

A medida que se van estandarizando los servidores de bases de datos, el factor diferenciador entre los productos de los diferentes fabricantes es el rendimiento de esos productos. Las **pruebas de rendimiento** son conjuntos de tareas que se emplean para cuantificar el rendimiento de los sistemas de software.

21.3.1. Familias de tareas

Dado que la mayor parte de los sistemas de software, como las bases de datos, son complejos hay bastante variación en su implementación por los diferentes fabricantes. En consecuencia, hay una variación significativa en su rendimiento en las diferentes tareas. Puede que un sistema sea el más eficiente en una tarea concreta y puede que otro lo sea en una tarea diferente. Por tanto, una sola tarea no suele resultar suficiente para cuantificar el rendimiento del sistema. En lugar de eso, el rendimiento de un sistema se mide mediante familias de tareas estandarizadas, denominadas *pruebas de rendimiento*.

La combinación de los resultados de rendimiento de varias tareas debe realizarse con cuidado. Supóngase que se tienen dos tareas, T_1 y T_2, y que se mide el flujo de un sistema como el número de transacciones de cada tipo que se ejecutan en un tiempo dado (digamos, un segundo). Supóngase que el sistema A ejecuta T_1 a 99 transacciones por segundo y que T_2 se ejecuta a una transacción por segundo. De manera parecida, supóngase que el sistema B ejecuta T_1 y T_2 a 50 transacciones por segundo. Supóngase también que una carga de trabajo tiene una mezcla a partes iguales de los dos tipos de transacciones.

Si se toma el promedio de los dos pares de resultados (es decir, 99 y 1 frente a 50 y 50), pudiera parecer que los dos sistemas tienen el mismo rendimiento. Sin embargo, sería *erróneo* tomar los promedios de esta manera (si se ejecutaran 50 transacciones de cada tipo el sistema *A* tardaría unos 50,5 segundos en concluirlas, mientras que el sistema *B* las terminaría ¡en sólo 2 segundos!).

Este ejemplo muestra que una sola medida del rendimiento induce a error si hay más de un tipo de transacción. El modo correcto de promediar los números es tomar el **tiempo para concluir** la carga de trabajo, en vez del **flujo** promedio de cada tipo de transacción. Se puede así calcular el rendimiento del sistema en transacciones por segundo para una carga de trabajo concreta con exactitud. Por tanto, el sistema A tarda 50,5/100, que son 0,505 segundos por transacción, mientras que el sistema B tarda 0,02 segundos por transacción, en promedio. En términos de flujo el sistema A se ejecuta a un promedio de 1,98 transacciones por segundo, mientras que el sistema B se ejecuta a 50 transacciones por segundo. Suponiendo que las transacciones de todos los tipos son igual de probables el modo correcto de promediar los flujos respecto de los diferentes tipos de transacciones es tomar la **media armónica** de los flujos. La media armónica de *n* flujos $f_1, \ldots, fn$ se define como

$$\frac{n}{\frac{1}{t_1} + \frac{1}{t_2} + \cdots + \frac{1}{t_n}}$$

Para este ejemplo la media armónica de los flujos en el sistema A es 1,98. Para el sistema B es 50. Por tanto, el sistema B es aproximadamente veinticinco veces más rápido que el sistema A para una carga de trabajo consistente en una mezcla a partes iguales de los dos tipos de transacciones de ejemplo.

21.3.2. Clases de aplicaciones de bases de datos

El **procesamiento en conexión de transacciones (Online Transaction Processing, OLTP**) y la **ayuda a la toma de decisiones** (incluyendo el **procesamiento en conexión analítico [Online Analytical Processing, OLAP]**) son dos grandes clases de aplicaciones manejadas por los sistemas de bases de datos. Estas dos clases de tareas tienen necesidades diferentes. La elevada concurrencia y las técnicas inteligentes para acelerar el procesamiento de las operaciones de compromiso se necesitan para soportar una elevada tasa de transacciones de actualización. Por otro lado, los buenos algoritmos para la evaluación de consultas y la optimización de las consultas son necesarios para la ayuda a la toma de decisiones. La arquitectura de algunos sistemas de bases de datos se ha ajustado para el procesamiento de las transacciones; la de otros, como la serie DBC de Teradata de sistemas paralelos de bases de datos, para el ayuda a la toma de decisiones. Otros fabricantes intentan conseguir un equilibrio entre las dos tareas.

Las aplicaciones suelen tener una mezcla de necesidades de procesamiento de transacciones y ayuda a la toma de decisiones. Por tanto, el mejor sistema de bases de datos para cada aplicación depende de la mezcla de las dos necesidades que tenga la aplicación.

Supóngase que se tienen resultados de flujo para las dos clases de aplicaciones por separado y la aplicación en cuestión tiene una mezcla de transacciones de las dos clases. Hay que ser precavido incluso al tomar la media armónica de los resultados de flujo, debido a la **interferencia** entre las transacciones. Por ejemplo, una transacción de ayuda a la toma de decisiones que tarde mucho en ejecutarse puede adquirir varios bloqueos, lo que puede evitar el progreso de las transacciones de actualización. La media armónica de los flujos sólo debe utilizarse si las transacciones no interfieren entre sí.

21.3.3. Las pruebas TPC

El **Consejo para el rendimiento del procesamiento de las transacciones (Transaction Processing Performance Council, TPC**) ha definido una serie de normas de índices para los sistemas de bases de datos. Los índices TPC se definen con gran minuciosidad. Definen el conjunto de relaciones y el tamaño de las tuplas. Definen el número de tuplas de las relaciones no como un número fijo, sino como un múltiplo del número de transacciones por segundo que se afirma que se realizan, para reflejar que una tasa mayor de ejecución de transacciones probablemente se halle correlacionada con un número mayor de cuentas. La métrica del rendimiento es el flujo, expresado como **transacciones por segundo (TPS)**. Cuando se mide el rendimiento, el sistema debe proporcionar un tiempo de respuesta que se halle dentro de ciertos límites, de modo que un flujo elevado no pueda obtenerse a expensas de tiempos de respuesta muy elevados. Además, para las aplicaciones profesionales, el coste es de gran importancia. Por tanto, el índice TPC también mide el rendimiento en términos de **precio por TPS**. Puede que los sistemas de gran tamaño tengan un elevado número de transacciones por segundo, pero puede que resulten caros (es decir, que tengan un precio elevado por TPS). Además, una compañía no puede afirmar que tiene resultados de los índices TPC en sus sistemas *sin* una auditoría externa que asegure que el sistema sigue fielmente la definición del índice, incluyendo el soporte pleno de las propiedades ACID de las transacciones.

El primero de la serie fue el **índice TPC-A**, que se definió en 1989. Este índice simula una aplicación bancaria típica mediante un solo tipo de transacción que modela la retirada y el depósito de efectivo en un cajero automático. La transacción actualiza varias relaciones (como el saldo del banco, el saldo del cajero y el saldo del cliente) y añade un registro a una relación de seguimiento para la auditoría. El índice también incorpora la comunicación con los terminales para modelar el rendimiento de extremo a extremo del sistema de manera realista. El **índice TPC-B** se diseñó para probar el rendimiento central del sistema de bases de datos, junto con el sistema operativo en el que se ejecuta el sistema. Elimina las partes del índice TPC-A que tratan de los usuarios, de las comunicaciones y de los terminales para centrarse en el servidor de bases de datos dorsal. Ni TPC-A ni TPC-B se emplean mucho hoy en día.

El **índice TPC-C** se diseñó para modelar un sistema más complejo que el del índice TPC-A. El índice TPC-C se concentra en las actividades principales de un entorno de admisión de pedidos, como son la entrada y la entrega de pedidos, el registro de los pagos, la verificación del estado de los pedidos y el seguimiento de los niveles de inventarios. El índice TPC-C se sigue utilizando con profusión para el procesamiento de transacciones.

El **índice TPC-D** se diseñó para probar el rendimiento de los sistemas de bases de datos en consultas de ayuda a la toma de decisiones. Los sistemas de ayuda a la toma de decisiones se están haciendo cada vez más importantes hoy en día. Los índices TPC-A, TPC-B y TPC-C miden el rendimiento de las cargas de procesamiento de transacciones y no deben utilizarse como medida del rendimiento en consultas de ayuda a la toma de decisiones. La D de TPC-D viene de **ayuda a la toma de Decisiones**. El esquema del índice TPC-D modela una aplicación de ventas/distribución, con componentes, clientes y pedidos, junto con cierta información auxiliar. El tamaño de las relaciones se define como una relación y el tamaño de la base de datos es el tamaño total de todas las relaciones, expresado en gigabytes. TPC-D con un factor de escala de uno representa el índice TPC-D para una base de datos de un gigabyte, mientras que el factor de escala diez representa una base de datos de diez gigabytes. La carga de trabajo del índice consiste en un conjunto de diecisiete consultas SQL que modelan tareas que se ejecutan frecuentemente en los sistemas de ayuda a la toma de decisiones. Parte de las consultas utilizan características complejas de SQL, como las consultas de agregación y las consultas anidadas.

Los usuarios de los índices se dieron cuenta pronto de que las diferentes consultas TPC-D podían acelerarse de manera significativa empleando las vistas materializadas y otra información redundante. Hay aplicaciones, como las tareas periódicas de información, donde las consultas se conocen con antelación y las vistas materializadas pueden seleccionarse con cuidado para acelerar las consultas. Resulta necesario, no obstante, tener en cuenta la sobrecarga que supone mantener las vistas materializadas.

El **índice TPC-R** (donde la R viene de **informar (Reporting)**) supone un refinamiento del índice TPC-D. El esquema es el mismo, pero hay veintidós consultas, de las cuales dieciséis provienen de TPC-D. Además, hay dos actualizaciones, un conjunto de inserciones y un conjunto de eliminaciones. Se permite que la base de datos que ejecuta el índice utilice vistas materializadas y otra información redundante.

A diferencia de esto, el **índice TPC-H** (donde la H representa **ad hoc**) utiliza el mismo esquema y la misma carga de trabajo que TPC-R, pero prohíbe las vistas materializadas y otra información redundante y permite los índices sólo en las claves principales y ajenas. Este índice modela consultas ad hoc donde las consultas no se conocen con anterioridad, por lo que no es posible crear con antelación vistas materializadas adecuadas.

Tanto TPC-H como TPC-R miden el rendimiento de esta manera: el **test de potencia** ejecuta las consultas y las actualizaciones una a una de manera secuencial y 3.600 segundos divididos por la media geométrica de los tiempos de ejecución de las consultas (en segundos) da una medida de las consultas por hora. El **test de flujo** ejecuta varias corrientes en paralelo, cada una de las cuales ejecuta las veintidós consultas. También hay una corriente de actualizaciones paralela. Aquí el tiempo total de toda la ejecución se utiliza para calcular el número de consultas por hora.

La **métrica compuesta consultas por hora**, que es la métrica global, se obtiene como la raíz cuadrada del producto de las métricas de potencia y de flujo. Se define una **métrica compuesta precio/rendimiento** dividiendo el precio del sistema por la métrica compuesta.

El índice de comercio Web **TPC-W** es un índice de extremo a extremo que modela los sitios Web que tienen contenido estático (imágenes, sobre todo) y contenido dinámico generado a partir de una base de datos. Se permite de manera explícita el almacenamiento en caché del contenido dinámico, dado que resulta muy útil para acelerar los sitios Web. El índice modela una librería electrónica, y como otros índices TPC, proporciona diferentes factores de escala. La métrica de rendimiento principal son las **interacciones Web por segundo (Web interactions per second, WIPS)** y el precio por WIPS.

21.3.4. Las pruebas BDOO

La naturaleza de las aplicaciones de las bases de datos orientadas a objetos (BDOO) es diferente de las de las aplicaciones típicas de procesamiento de transacciones. Por tanto, se ha propuesto un conjunto diferente de índices para las BDOO. La prueba operaciones con objetos, versión 1, popularmente conocido como **índice OO1**, fue una de las primeras propuestas. El **índice OO7** sigue una filosofía diferente de la de los índices TPC. Los índices TPC proporcionan uno o dos resultados (en términos del promedio de transacciones por segundo y de transacciones por segundo y por dólar); el índice OO7 proporciona un conjunto de resultados, que contienen un resultado de índice independiente para cada una de las diferentes clases de operaciones. El motivo de este enfoque es que no está todavía claro lo que es la transacción BDOO *típica*. Está claro que esa transacción llevará a cabo ciertas operaciones, como recorrer un conjunto de objetos conectados o recuperar todos los objetos de una clase, pero no está claro exactamente la mezcla de tales operaciones que se utilizará. Por tanto, el índice proporciona resultados separados para cada clase de operaciones; los resultados pueden combinarse de la manera adecuada, en función de la aplicación concreta.

21.4. NORMALIZACIÓN

Las **normas** definen las interfaces de los sistemas de software; por ejemplo, las normas definen la sintaxis y la semántica de los lenguajes de programación o las funciones en la interfaz de los programas de aplicaciones o, incluso, los modelos de datos (como las normas de las bases de datos orientadas a los objetos). Hoy en día los sistemas de bases de datos son complejos y suelen estar constituidos por varias partes creadas de manera independiente que deben interactuar entre sí. Por ejemplo, puede que los programas clientes se creen de manera independiente de los sistemas dorsales, pero todos ellos deben poder interactuar entre sí. Puede que una empresa que tenga varios sistemas de bases de datos heterogéneos necesite intercambiar datos entre las bases de datos. En una situación de este tipo las normas desempeñan un papel importante.

Las **normas formales** son las que han sido desarrolladas por una organización de normalización o por grupos de empresas mediante un procedimiento público. Los productos dominantes se convierten a veces en una **norma de facto**, en el sentido de que resultan aceptados de manera general como normas sin necesidad de ningún procedimiento formal de reconocimiento. Algunas normas formales, como muchos aspectos de las normas SQL-92 y SQL:1999, son **normas anticipativas** que lideran el mercado; definen las características que los fabricantes implementan posteriormente en los productos. En otros casos, las normas, o partes de las normas, son **normas reaccionarias**, en el sentido de que intentan normalizar las características que ya han implementado algunos fabricantes, y que pueden haberse convertido, incluso, en normas de facto. SQL-89 era, en muchos sentidos, reaccionario, dado que estandarizaba características, como la comprobación de la integridad, que ya estaban presentes en la norma SAA SQL de IBM y en otras bases de datos.

Los comités para las normas formales suelen estar compuestos por representantes de los fabricantes y por miembros de grupos de usuarios y de organizaciones de normalización como la Organización internacional de

normalización (International Organization for Standardization, ISO) o el Instituto nacional americano de normalización (American National Standards Institute, ANSI) o de organismos profesionales como el Instituto de ingenieros eléctricos y electrónicos (Institute of Electrical and Electronics Engineers, IEEE). Los comités para las normas formales se reúnen de manera periódica y sus componentes presentan propuestas de características de la norma que hay que añadir o modificar. Tras un periodo de discusión (generalmente amplio), de modificaciones de la propuesta y de examen público, los integrantes de los comités votan la aceptación o el rechazo de las características. Cierto tiempo después de la definición e implementación de una norma quedan claras sus carencias y aparecen nuevas necesidades. El proceso de actualización de la norma comienza entonces y tras unos cuantos años suele publicarse una nueva versión de la misma. Este ciclo suele repetirse cada pocos años hasta que finalmente (quizás muchos años más tarde) la norma se vuelve tecnológicamente irrelevante o pierde su base de usuarios.

La norma CODASYL de DBTG para bases de datos en red, formulada por el Grupo de trabajo para bases de datos (Database Task Group, DBTG), fue una de las primeras normas formales en este campo. Los productos de bases de datos de IBM solían establecer las normas de facto, dado que IBM ocupaba gran parte de este mercado. Con el crecimiento de las bases de datos relacionales aparecieron nuevos competidores en el negocio de las bases de datos; por tanto, surgió la necesidad de normas formales. En los últimos años Microsoft ha creado varias especificaciones que también se han convertido en normas de facto. Un ejemplo destacable es ODBC, que se utiliza actualmente en entornos que no son de Microsoft. JDBC, cuya especificación fue creada por Sun Microsystems, es otra norma de facto muy utilizada.

Este apartado ofrece una introducción de muy alto nivel a las diferentes normas, concentrándose en los objetivos de cada norma. Las notas bibliográficas al final del capítulo ofrecen referencias de las descripciones detalladas de las normas mencionados en este apartado.

21.4.1. Normas de SQL

Dado que SQL es el lenguaje de consultas más utilizado se ha trabajado mucho en su normalización. ANSI e ISO, con los diferentes fabricantes de bases de datos, han desempeñado un papel protagonista en esta labor. La norma SQL-86 fue la versión inicial. La norma Arquitectura de aplicaciones de sistemas (Systems Application Architecture, SAA) de IBM para SQL se publicó en 1987. A medida que la gente identificaba la necesidad de más características se desarrollaron versiones actualizadas de la norma formal de SQL, denominadas SQL-89 y SQL-92.

La última versión de la norma SQL, denominada SQL:1999, añade varias características al lenguaje. Ya se han visto muchas de estas características en los capítulos anteriores, y se verán unas cuantas más en los siguientes. La norma se ha dividido en varias partes:

- SQL/Framework (Parte 1) proporciona una introducción la norma.
- SQL/Foundation (Parte 2) define los fundamentos de la norma: tipos, esquemas, tablas, vistas, consultas y sentencias de actualización, expresiones, modelo de seguridad, predicados, reglas de asignación, gestión de transacciones, etcétera.
- SQL/CLI (Call Level Interface) (Parte 3) define las interfaces de los programas de aplicaciones con SQL.
- SQL/PSM (Persistent Stored Modules) (Parte 4) define las extensiones de SQL para hacerlo procedimental.
- SQL/Bindings (Parte 5) define las normas para SQL incorporado para diferentes lenguajes.

Las características OLAP de SQL:1999 (Apartado 22.2.3) se han especificado como enmienda a la versión anterior de la norma SQL:1999. Hay otras partes en desarrollo, incluyendo

- Parte 7: SQL/Temporal aborda las normas para los datos temporales.
- Parte 9: SQL/MED (Management of External Data) define las normas para la realización de interfaces en los sistemas SQL con orígenes externos. Al escribir envolturas, los diseñadores de los sistemas pueden tratar los orígenes externos de datos, como pueden ser los archivos o los datos de bases de datos no relacionales, como si fueran tablas *externas*.
- Parte 10: SQL/OLB (Object Language Bindings) define las normas para la incrustación de SQL en Java.

Los números que faltan (partes 6 y 8) tratan características como el procesamiento de transacciones distribuidas y los datos multimedia para las que todavía no hay ningún acuerdo sobre las normas. Las normas multimedia pretenden tratar el almacenamiento y la recuperación de datos de texto, datos espaciales e imágenes fijas.

21.4.2. Las normas de conectividad de las bases de datos

La norma **ODBC** es una norma muy utilizada para la comunicación entre las aplicaciones clientes y los sistemas de bases de datos. ODBC se basa en las normas SQL **Interfaz de nivel de llamada (Call-Level Interface, CLI)** desarrolladas por el consorcio industrial *X/Open* y el Grupo SQL Access, pero tienen varias extensiones. La API ODBC define una CLI, una definición de sintaxis SQL y reglas sobre las secuencias admisibles de llamadas CLI. La norma también define los niveles de conformidad para la CLI y la sintaxis

SQL. Por ejemplo, el nivel central de la CLI tiene comandos para conectarse con bases de datos, para preparar y ejecutar sentencias SQL, para devolver resultados o valores de estado y para administrar transacciones. El siguiente nivel de conformidad (nivel uno) exige el soporte de la recuperación de información de los catálogos y otras características que superan la CLI del nivel central; el nivel dos exige más características, como la capacidad de enviar y recuperar arrays de valores de parámetros y de recuperar información de catálogo más detallada.

ODBC permite que un cliente se conecte de manera simultánea con varios orígenes de datos y que conmute entre ellos, pero las transacciones en cada uno de ellos son independientes entre sí; ODBC no soporta el compromiso de dos fases.

Los sistemas distribuidos ofrecen un entorno más general que los sistemas cliente-servidor. El consorcio X/Open también ha desarrollado las normas **X/Open XA** para la interoperación de las bases de datos. Estas normas definen las primitivas de gestión de las transacciones (como pueden ser el comienzo de las transacciones, su compromiso, su anulación y su preparación para el compromiso) que deben proporcionar las bases de datos que cumplan con la norma; los administradores de las transacciones pueden invocar estas primitivas para implementar las transacciones distribuidas mediante compromiso de dos fases. Las normas XA son independientes de los modelos de datos y de las interfaces concretas entre los clientes y las bases de datos para el intercambio de datos. Por tanto, se pueden utilizar los protocolos XA para implementar sistemas de transacciones distribuidas en los que una sola transacción pueda tener acceso a bases de datos relacionales y orientadas a objetos y que el administrador de transacciones asegure la consistencia global mediante el compromiso de dos fases.

Hay muchos orígenes de datos que no son bases de datos relacionales y, de hecho, puede que no sean ni siquiera bases de datos. Ejemplos de esto son los archivos planos y los almacenes de correo electrónico. **OLE-DB** de Microsoft es una API de C++ con objetivos parecidos a los de ODBC, pero para orígenes de datos que no son bases de datos y que puede que sólo proporcionen servicios limitados de consulta y de actualización. Al igual que ODBC, OLE-DB proporciona estructuras para la conexión con orígenes de datos, el inicio de una sesión, la ejecución de comandos y la devolución de resultados en forma de conjunto de filas, que es un conjunto de filas de resultados.

Sin embargo, OLE-DB se diferencia de ODBC en varios aspectos. Para dar soporte a los orígenes de datos con soporte de características limitadas, las características de OLE-DB se dividen entre varias interfaces y cada origen de datos sólo puede implementar un subconjunto de esas interfaces. Los programas OLE-DB pueden negociar con los orígenes de datos para averiguar las interfaces que soportan. En ODBC los comandos siempre están en SQL. En OLE-DB, los comandos pueden estar en cualquier lenguaje soportado por el origen de datos; aunque puede que algunos orígenes de datos soporten SQL, o un subconjunto limitado de SQL, puede que otros orígenes sólo ofrezcan posibilidades sencillas como el acceso a los datos de los archivos planos, sin ninguna capacidad de consulta. Otra diferencia importante de OLE-DB con ODBC es que los conjuntos de filas son objetos que pueden compartir varias aplicaciones mediante la memoria compartida. Los objetos conjuntos de filas los puede actualizar una aplicación, y a las otras aplicaciones que compartan ese objeto se les notificará la modificación.

La API **Objetos activos de datos (Active Data Objects, ADO)**, también creada por Microsoft, ofrece una interfaz sencilla de utilizar con la funcionalidad OLE-DB, que puede llamarse desde los lenguajes de guiones, como VBScript y JScript.

21.4.3. Normas de las bases de datos de objetos

Hasta ahora las normas en el área de las bases de datos orientadas a objetos los han impulsado sobre todo los fabricantes de BDOO. El *Grupo de gestión de bases de datos de objetos (Object Database Management Group,* ODMG) es un grupo formado por fabricantes de BDOO para normalizar los modelos de datos y las interfaces de lenguaje con las BDOOs. La interfaz del lenguaje C++ especificada por ODMG se estudió en el Capítulo 8. ODMG también ha especificado una interfaz Java y una interfaz Smalltalk.

El *Grupo de administración de objetos (Object Management Group,* OMG) es un consorcio de empresas, formado con el objetivo de desarrollar una arquitectura estándar para las aplicaciones de software distribuido basadas en el modelo orientado a objetos. OMG creó el modelo de referencia *Arquitectura de gestión de objetos (Object Management Architecture,* OMA). El *Agente para solicitudes de objetos (Object Request Broker,* ORB) es un componente de la arquitectura OMA que proporciona de manera transparente la entrega de mensajes a los objetos distribuidos, de modo que la ubicación física del objeto no tiene importancia. La **Arquitectura común de agente para solicitudes de objetos (Common Object Request Broker Architecture, CORBA**) proporciona una especificación detallada del ORB, e incluye un **Lenguaje de descripción de interfaces (Interface Description Language, IDL**), que se utiliza para definir los tipos de datos empleados para el intercambio de datos. El IDL ayuda a dar soporte a la conversión de datos cuando se intercambian entre sistemas con diferentes representaciones de los datos.

21.4.4. Normas basadas en XML

Se ha definido una amplia variedad de normas basadas en XML (véase el Capítulo 10) para gran número de aplicaciones. Muchas de estas normas están relacionadas con

el comercio electrónico. Entre ellas hay normas promulgados por consorcios sin ánimo de lucro y esfuerzos apoyados por las empresas para crear normas de facto. RosettaNet, que pertenece a la primera categoría, utiliza normas basadas en XML para facilitar la gestión de las cadenas de abastecimiento en las industrias informáticas y de tecnologías de la información. Empresas como Commerce One proporcionan sistemas de adquisición basados en Web, gestión de la cadena de abastecimiento y mercados electrónicos (incluidas las subastas en conexión). BizTalk es una infraestructura de esquemas y directrices XML, apoyada por Microsoft. Estos y otros marcos definen catálogos, descripciones de servicios, facturas, órdenes de compra, solicitudes de estado de pedidos, albaranes de envío y elementos relacionados.

Los participantes en los mercados electrónicos pueden guardar los datos en gran variedad de sistemas de bases de datos. Puede que estos sistemas utilicen diferentes modelos, formatos y tipos de datos. Además, puede que haya diferencias semánticas (sistema métrico frente a sistema imperial británico, distintas divisas, etcétera) en los datos. Las normas para los mercados electrónicos incluyen los métodos para *envolver* cada uno de estos sistemas heterogéneos con un esquema XML. Estas *envolturas* XML forman la base de una vista unificada de los datos para todos los participantes del mercado.

El *protocolo simple de acceso a objetos (Simple Object Access Protocol,* SOAP) es una norma para llamadas a procedimientos remotos que utiliza XML para codificar los datos (tanto los parámetros como los resultados) y utiliza HTTP como protocolo de transporte; es decir, las llamadas a procedimientos se transforman en solicitudes HTTP. SOAP está apoyado por el Consorcio World Wide Web (World Wide Web Consortium, W3C) y está logrando una amplia aceptación en la industria (incluidos IBM y Microsoft). SOAP puede utilizarse en gran variedad de aplicaciones. Por ejemplo, en el comercio electrónico entre empresas, las aplicaciones que se ejecutan en un sitio pueden tener acceso a los datos de otros sitios mediante SOAP. Microsoft ha definido las normas para el acceso OLAP y para la búsqueda de datos con SOAP (OLAP y la búsqueda de datos se tratan en el Capítulo 22).

El lenguaje estándar de consultas de W3C para XML se denomina *XQuery*. Ya en 2001 la norma se hallaba en la etapa de pruebas de funcionamiento, y debía estar finalizado para finales de ese año. Entre los lenguajes de consulta XML anteriores están *Quilt* (en el que se basa XQuery), *XML-QL* y *XQL*.

21.5. COMERCIO ELECTRÓNICO**

El término comercio electrónico hace referencia al proceso de llevar a cabo varias actividades relacionadas con el comercio por medios electrónicos, principalmente por Internet. Entre los tipos de actividades figuran:

- Actividades previas a la venta, necesarias para informar al posible comprador del producto o servicio que se desea vender.
- El proceso de venta, que incluye las negociaciones sobre el precio y la calidad del servicio, y otros asuntos contractuales.
- El mercado: cuando hay varios vendedores y varios compradores para un mismo producto, un mercado, como la bolsa, ayuda a negociar el precio que se va a pagar por el producto. Las subastas se utilizan cuando hay un único vendedor y varios compradores, y las subastas inversas se emplean cuando hay un solo comprador y varios vendedores.
- El pago de la compra.
- Las actividades relacionadas con la entrega del producto o servicio. Algunos productos y servicios pueden entregarse por Internet; para otros, Internet sólo se utiliza para facilitar información sobre el envío y para realizar un seguimiento de los envíos de los productos.
- Soporte al cliente y servicio postventa.

Las bases de datos se utilizan ampliamente para soportar estas actividades. Para algunas de las actividades el empleo de las bases de datos es directo, pero hay aspectos interesantes de desarrollo de aplicaciones para las demás actividades.

21.5.1. Catálogos electrónicos

Cualquier sitio de comercio electrónico proporciona a los usuarios un catálogo de los productos y servicios que ofrece. Los servicios facilitados por los catálogos electrónicos pueden variar considerablemente.

Como mínimo, un catálogo electrónico debe proporcionar servicios de exploración y de búsqueda para ayudar a los clientes a hallar el producto que buscan. Para ayudar en la exploración conviene que los productos estén organizados en una jerarquía intuitiva, de modo que unas pocas pulsaciones en los hipervínculos puedan llevar a los clientes a los productos en los que estén interesados. Las palabras clave facilitadas por el cliente (por ejemplo, «cámara digital» o «computadora») deben acelerar el proceso de búsqueda de los productos solicitados. Los catálogos electrónicos también deben proporcionar un medio para que los clientes comparen con facilidad las alternativas para elegir entre los diferentes productos.

Los catálogos electrónicos pueden personalizarse para los clientes. Por ejemplo, puede que un detallista

tenga un acuerdo con una gran empresa para venderle algunos productos con descuento. Un empleado de la empresa, al examinar el catálogo para adquirir productos para la empresa, debería ver los precios con el descuento acordado, en vez de con los precios normales. Debido a las restricciones legales sobre las ventas de algunos tipos de artículos, no se les deberían mostrar a los clientes menores de edad, o de ciertos estados o países, los artículos que no se les pueden vender legalmente. Los catálogos también pueden personalizarse para usuarios individuales, de acuerdo con su historial de compras. Por ejemplo, se pueden ofrecer a los clientes frecuentes descuentos especiales en algunos productos.

El soporte de esa personalización necesita que la información de los clientes y la información sobre precios o descuentos especiales y sobre restricciones a las ventas se guarden en una base de datos. También hay desafíos en el soporte de tasas de transacciones muy elevadas, que suelen abordarse guardando en la caché los resultados de las consultas o las páginas Web generadas.

21.5.2. Mercados

Cuando hay varios vendedores o varios compradores (o ambas cosas) para un producto, los mercados ayudan a negociar el precio que debe pagarse por el producto. Hay varios tipos diferentes de mercados:

- En los sistemas de **subastas inversas** los compradores manifiestan sus necesidades y los vendedores pujan por proporcionar el artículo. El proveedor que ofrece el precio más bajo gana la subasta. En los sistemas de puja cerrada las pujas no se hacen públicas, mientras que en los sistemas de puja abierta las pujas sí se hacen públicas.
- En las **subastas** hay varios compradores y un solo vendedor. En aras de la sencillez, supóngase que sólo se vende un ejemplar de cada artículo. Los compradores pujan por los artículos que se venden y el que puja más alto por un artículo consigue comprarlo por el precio de la puja.

  Cuando hay varios ejemplares de un artículo las cosas se vuelven más complicadas: supóngase que hay cuatro artículos y puede que un comprador desee tres ejemplares a 10 € cada uno, mientras que otro desea dos copias por 13 € cada una. No es posible satisfacer ambas pujas. Si los artículos carecen de valor si no se venden (por ejemplo, billetes de avión, que deben venderse antes de que despegue el avión), el vendedor simplemente selecciona un conjunto de pujas que maximice los ingresos. En caso contrario, la decisión es más complicada.
- En las **bolsas**, como puede ser una bolsa de valores, hay varios vendedores y varios compradores. Los compradores pueden especificar el precio máximo que desean pagar, mientras que los vendedores especifican el precio mínimo que desean. Suele haber un *creador de mercado* que aúna las ofertas de compra y de venta y decide el precio de cada intercambio (por ejemplo, al precio de la oferta de venta).

  Hay otros tipos de mercados más complejos.

Entre los aspectos de las bases de datos relacionados con el manejo de los mercados figuran:

- Hay que autentificar a los que realizan las pujas antes de permitirles pujar.
- Las pujas (de compra o de venta) deben registrarse de modo seguro en una base de datos. Hay que comunicar rápidamente las pujas al resto de las personas implicadas en el mercado (como pueden ser todos los compradores o todos los vendedores), que puede que sean numerosas.
- Los retrasos en la difusión de las pujas pueden llevar a pérdidas financieras para algunos participantes.
- Los volúmenes de los intercambios pueden resultar tremendamente grandes en tiempos de volatilidad de las bolsas, o hacia el final de las subastas. Por tanto, se utilizan para estos sistemas bases de datos de muy alto rendimiento con elevados grados de paralelismo.

21.5.3. Liquidación de pedidos

Una vez seleccionados los artículos (quizás mediante un catálogo electrónico) y determinado el precio (acaso mediante un mercado electrónico), hay que liquidar el pedido. La liquidación implica el pago y la entrega de las mercancías.

Una manera sencilla pero poco segura de pagar electrónicamente consiste en enviar el número de una tarjeta de crédito. Hay dos problemas principales. En primer lugar, es posible el fraude con las tarjetas de crédito. Cuando un comprador paga mercancías físicas las empresas pueden asegurarse de que la dirección de entrega coincide con la del titular de la tarjeta, de modo que nadie más reciba la mercancía, pero para las mercan-cías entregadas de manera electrónica no es posible realizar esa comprobación. En segundo lugar, hay que confiar en que el vendedor sólo facture el artículo acordado y en que no pase el número de la tarjeta de crédito a personas no autorizadas que lo puedan utilizar de manera no adecuada.

Se dispone de varios protocolos para los pagos seguros que evitan los dos problemas mencionados. Además, proporcionan una mayor intimidad, ya que no hay que dar al vendedor datos innecesarios del comprador y no se le facilitan a la compañía de la tarjeta de crédito información innecesaria de los artículos comprados. Toda la información transmitida debe cifrarse de modo que nadie que intercepte los datos por la red pueda averiguar su contenido. El cifrado con claves pública y privada se utiliza mucho para esta tarea.

Los protocolos también deben evitar los **ataques de personas intermedias**, en los que alguien puede suplantar al banco o a la compañía de la tarjeta de crédito, o incluso al vendedor o al comprador y sustraer información secreta. La suplantación puede realizarse pasando una clave falsa como si fuera la clave pública de otra persona (el banco, la compañía de la tarjeta de crédito, el comerciante o el comprador). La suplantación se evita mediante un sistema de **certificados digitales**, en el que las claves públicas vienen firmadas por una agencia de certificación cuya clave pública es bien conocida (o que, a su vez, tiene su clave pública certificada por otra agencia de certificación, y así hasta llegar a una clave que sea bien conocida). A partir de la clave pública bien conocida el sistema puede autentificar las otras claves comprobando los certificados en una secuencia inversa.

El protocolo **transacciones electrónicas seguras** (**Secure Electronic Transaction, SET**) es uno de los protocolos de pago seguro. El protocolo necesita varias rondas de comunicación entre el comprador, el vendedor y el banco para garantizar la seguridad de la transacción.

También hay sistemas que ofrecen más anonimato, parecido al proporcionado por el dinero físico en efectivo. El sistema de pagos **DigiCash** es uno de esos sistemas. Cuando se realiza un pago en estos sistemas no es posible identificar al comprador. Sin embargo, la identificación del comprador resulta muy sencilla con las tarjetas de crédito, e incluso en el caso de SET es posible identificar al comprador con la colaboración de la compañía emisora de la tarjeta de crédito o del banco.

21.6. SISTEMAS HEREDADOS

Los **sistemas heredados** son sistemas de generaciones anteriores que son incompatibles con las normas y sistemas de la generación actual. Puede que estos sistemas todavía contengan datos valiosos y den soporte a aplicaciones críticas. Los sistemas heredados de hoy en día suelen ser los construidos con tecnologías como las bases de datos que utilizan los modelos de datos de red o jerárquico o utilizan Cobol y sistemas de archivos sin bases de datos.

El traspaso de las aplicaciones heredadas a entornos más modernos suele resultar costoso tanto en términos de tiempo como de dinero, dado que suelen ser de tamaño muy grande, con millones de líneas de código desarrolladas por equipos de programadores a lo largo de varias décadas.

Por eso es importante dar soporte a estos sistemas de generaciones anteriores, o heredados, y facilitar su interoperatividad con los sistemas más modernos. Un enfoque empleado para interoperar entre las bases de datos relacionales y las bases de datos heredadas es crear una capa, denominada **envoltura**, por encima de los sistemas heredados que pueda hacer que el sistema heredado parezca una bases de datos relacional. Puede que la envoltura ofrezca soporte para ODBC u otras normas de interconexión como OLE-DB, que puede utilizarse para consultar y actualizar el sistema heredado. La envoltura es responsable de la conversión de las consultas y actualizaciones relacionales en consultas y actualizaciones del sistema heredado.

Cuando una organización decide sustituir un sistema heredado por otro nuevo debe seguir un proceso denominado **ingeniería inversa**, que consiste en repasar el código del sistema heredado para obtener el diseño de los esquemas del modelo de datos requerido (como puede ser el modelo E-R o un modelo de datos orientado a los objetos). La ingeniería inversa también examina el código para averiguar los procedimientos y los procesos que se implementan con objeto de obtener un modelo de alto nivel del sistema. La ingeniería inversa es necesaria porque los sistemas heredados no suelen tener documentación de alto nivel de sus esquemas ni del diseño global de sus sistemas. Al presentarse el diseño de un nuevo sistema se repasa el diseño de modo que pueda mejorarse en lugar de volver a implementarlo tal como estaba. Se necesita una amplia labor de codificación para dar soporte a toda la funcionalidad (como pueden ser las interfaces de usuario y los sistemas de información) que proporcionaba el sistema heredado. El proceso completo se denomina **reingeniería**.

Cuando se ha creado y probado el sistema nuevo hay que llenarlo con los datos del sistema heredado y todas las actividades posteriores se deben realizar en el sistema nuevo. No obstante, la transición abrupta al sistema nuevo, que se denomina **enfoque gran explosión**, conlleva varios riesgos. En primer lugar, puede que los usuarios no estén familiarizados con las interfaces del sistema nuevo. En segundo lugar, puede haber fallos o problemas de rendimiento en el sistema nuevo que no se hayan descubierto al probarlo. Estos problemas pueden provocar grandes pérdidas a las empresas, dado que puede resultar gravemente afectada su capacidad para realizar transacciones críticas como las compras y las ventas. En algunos casos extremos se ha llegado a abandonar el sistema nuevo y se ha vuelto a utilizar el sistema heredado después de que fallara el intento de cambio.

Un enfoque alternativo, denominado **enfoque incremental**, sustituye la funcionalidad del sistema heredado de manera incremental. Por ejemplo, puede que las nuevas interfaces de usuario se utilicen con el sistema antiguo en el dorsal, o viceversa. Otra opción es utilizar el sistema nuevo sólo para algunas funcionali-

dades que puedan desgajarse del sistema heredado. En cualquier caso, los sistemas heredados y los nuevos coexisten durante algún tiempo. Por tanto, existe una necesidad de desarrollo y empleo de envolturas del sistema heredado para proporcionar la funcionalidad requerida para interoperar con el sistema nuevo. Este enfoque, por tanto, tiene asociado un coste de desarrollo superior.

21.7. RESUMEN

- Los exploradores Web se han constituido en la interfaz de usuario con las bases de datos más utilizada. HTML proporciona la posibilidad de definir las interfaces que combinan los hipervínculos con los formularios. Los exploradores Web se comunican con los servidores Web mediante el protocolo HTTP. Los servidores Web pueden pasar las solicitudes a los programas de las aplicaciones y devolver los resultados al explorador.
- Hay varios lenguajes de guiones del lado del cliente (Javascript es el más utilizado) que proporcionan una mayor interacción con el usuario en el extremo del explorador.
- Los servidores Web ejecutan los programas de las aplicaciones para implementar la funcionalidad deseada. Las servlets son un mecanismo muy utilizado para escribir programas de aplicaciones que se ejecutan como parte del proceso del servidor Web para reducir las sobrecargas. También hay muchos lenguajes de guiones del lado del servidor que interpreta el servidor Web y proporcionan la funcionalidad de los programas de aplicaciones como parte del servidor Web.
- El ajuste de los parámetros del sistema de bases de datos, así como el diseño de nivel superior de la base de datos (como pueden ser el esquema, los índices y las transacciones) resultan importantes para lograr un buen rendimiento. La mejor forma de realizar el ajuste es identificar los cuellos de botella y eliminarlos.
- Las pruebas de rendimiento desempeñan un papel importante en las comparaciones entre sistemas de bases de datos, especialmente a medida que cumplen cada vez más las normas. El conjunto de pruebas TPC se utiliza mucho y las diferentes pruebas TPC resultan útiles para la comparación del rendimiento de las bases de datos con diferentes cargas de trabajo.
- Las normas son importantes debido a la complejidad de los sistemas de bases de datos y a la necesidad de interoperatividad. Hay normas formales para SQL. Las normas de facto, como ODBC y JDBC, y las normas adoptadas por grupos de empresas, como CORBA, han desempeñado un papel importante en el crecimiento de los sistemas de bases de datos cliente-servidor. Hay grupos de empresas desarrollando normas para las bases de datos orientadas a objetos, como ODMG.
- Los sistemas de comercio electrónico se están convirtiendo con rapidez en la parte principal del modo en que se realiza el comercio. Hay varios aspectos de las bases de datos en los sistemas de comercio electrónico. La administración de los catálogos, especialmente su personalización, se realiza con bases de datos. Los mercados electrónicos ayudan a fijar el precio de los productos mediante subastas, subastas inversas o bolsas. Se necesitan sistemas de bases de datos de alto rendimiento para manejar este intercambio. Los pedidos se liquidan mediante sistemas de pago electrónico, que también necesitan sistemas de bases de datos de alto rendimiento para manejar tasas de transacciones muy elevadas.
- Los sistemas heredados son sistemas basados en tecnologías de generaciones anteriores como las bases de datos no relacionales o, incluso, directamente en sistemas de archivos. Suele ser importante el establecimiento de las interfaces entre los sistemas heredados y los sistemas de última generación cuando ejecutan sistemas de misión crítica. La migración desde los sistemas heredados a los sistemas de última generación debe realizarse con cuidado para evitar interrupciones, que pueden resultar muy costosas.

TÉRMINOS DE REPASO

- Ajuste del esquema
- Ajuste del hardware
- Ajuste de los índices
- Ajuste del rendimiento
- Ajuste de las transacciones
- *Applets*
- Catálogos electrónicos
- Certificados digitales
- Clases de aplicaciones de bases de datos
- Cookie
- Comercio electrónico
- Cuellos de botella

- Guiones del lado cliente
- Guiones del lado del servidor
- Hipervínculos
- Ingeniería inversa
- Interacciones Web por segundo
- Interfaces Web con las bases de datos
- Interfaz de pasarela común (Common Gateway Interface, CGI)
- Lenguaje de guiones del lado del cliente
- Lenguaje de marcas de hipertexto (HyperTextMarkup Language, HTML)
- Liquidación de pedidos
- Localizador uniforme de recursos (Uniform resource locator, URL)
- Mantenimiento diferido de vistas
- Mantenimiento inmediato de vistas
- Mejora de la orientación del conjunto
- Mercados
  — Bolsas
  — Subastas
  — Subastas inversas
- Normalización
  — Normas anticipativas
  — Normas de facto
  — Normas formales
  — Normas reaccionarias
- Normas basadas en XML
- Normas de bases de datos de objetos
  — CORBA
  — ODMG
- Normas de conectividad para bases de datos
  — Normas X/Open XA
  — ODBC
  — OLE-DB
- Parámetros ajustables
- Procesamiento de transacciones por minilotes
- Protocolo de transferencia de hipertexto (HyperText Transfer Protocol, HTTP)
- Pruebas de BDOO
  — OO1
  — OO7
- Pruebas de rendimiento
- Pruebas TPC
  — TPC-A
  — TPC-B
  — TPC-C
  — TPC-D
  — TPC-R
  — TPC-H
  — TPC-W
- Regla de los cinco minutos
- Regla del minuto
- Reingeniería
- Servidores Web
- Servlets
- Sesión
- Simulación del rendimiento
- Sin conexión
- Sistemas de colas
- Sistemas heredados
- Tiempo hasta su finalización
- Tiempo de servicio
- Vistas materializadas

EJERCICIOS

21.1 ¿Cuál es la razón principal por la que los servlets dan mejor rendimiento que los programas que utilizan la interfaz de pasarela común (common gateway interface, CGI), pese a que los programas Java suelen ejecutarse más lentamente que los programas C o C++?

21.2 Indíquense algunas de las ventajas y de los inconvenientes de los protocolos sin conexión frente a los protocolos que mantienen las conexiones.

21.3 Indíquense tres maneras en que se puede utilizar el almacenamiento en caché para acelerar el rendimiento de los servidores Web.

21.4 **a.** ¿Cuáles son los tres niveles principales en los que se puede ajustar un sistema de bases de datos para mejorar su rendimiento?

b. Dense dos ejemplos del modo en que se puede realizar el ajuste para cada uno de los niveles.

21.5 ¿Cuál es el motivo para separar una transacción de larga duración en una serie de transacciones más breves? ¿Qué problemas pueden surgir como consecuencia y cómo pueden evitarse?

21.6 Supóngase que un sistema ejecuta tres tipos de transacciones. Las transacciones de tipo A se ejecutan a razón de 50 por segundo, las transacciones de tipo B

se ejecutan a 100 por segundo y las transacciones de tipo C se ejecutan a 200 por segundo. Supóngase que la mezcla de transacciones tiene un 25 por ciento del tipo A, otro 25 por ciento del tipo B y un 50 por ciento del tipo C.

a. ¿Cuál es el flujo promedio de transacciones del sistema, suponiendo que no hay interferencia entre las transacciones?

b. ¿Qué factores pueden generar interferencias entre las transacciones de los diferentes tipos, haciendo que el flujo calculado sea incorrecto?

21.7 Supóngase que el precio de la memoria cae a la mitad y la velocidad de acceso al disco (número de accesos por segundo) se dobla mientras el resto de los factores permanecen iguales. ¿Cuál sería el efecto de este cambio en las reglas de los cinco minutos y del minuto?

21.8 Indíquense algunas de las características de las pruebas TPC que ayudan a hacerlas medidas realistas y dignas de confianza.

21.9 ¿Por qué se sustituyó al índice TPC-D por los pruebas TPC-H y TPC-R?

21.10 Indíquense algunas ventajas e inconvenientes de las normas anticipativas frente a las normas reaccionarias.

21.11 Supóngase que alguien suplanta a una empresa y obtiene un certificado de una autoridad emisora de certificados. ¿Cuál es el efecto sobre las cosas (como las órdenes de compra o los programas) certificadas por la empresa suplantada y sobre las cosas certificadas por otras empresas?

SUGERENCIAS DE PROYECTOS

Cada uno de los que aparecen a continuación es un proyecto de grandes dimensiones, que puede ser un proyecto de un semestre de duración realizado por un grupo de estudiantes. La dificultad de cada proyecto puede ajustarse fácilmente añadiendo o eliminando características.

Proyecto 21.1 Considérese el esquema E-R del Ejercicio 2.7 (Capítulo 2), que representa la información de los equipos de una liga. Diséñese e impleméntese un sistema basado en la Web para introducir, actualizar y examinar los datos.

Proyecto 21.2 Diséñese e impleméntese un sistema de carro de la compra que permita a los compradores reunir artículos en un carro de la compra (se puede decidir la información que se proporciona de cada artículo) y comprarlos todos juntos. Se puede extender y utilizar el esquema E-R del Ejercicio 2.12 del Capítulo 2. Conviene comprobar la disponibilidad del artículo y tratar los artículos no disponibles del modo que se considere adecuado.

Proyecto 21.3 Diséñese e impleméntese un sistema basado en la Web para registrar la información sobre las matrículas y los cursos de los estudiantes para los cursos de una universidad.

Proyecto 21.4 Diséñese e impleméntese un sistema que permita el registro de la información de rendimiento de los cursos (concretamente, las notas dadas a cada estudiante en cada trabajo o examen de cada curso, y el cálculo de una suma [ponderada] de las notas para obtener las notas totales del curso). El número de trabajos o exámenes no debe estar predefinido; es decir, se pueden añadir más trabajos o exámenes en cualquier momento. El sistema también debe soportar la separación, permitiendo que se especifiquen separadores para varios niveles.

Puede que también se desee integrarlo con el sistema de matrícula de los estudiantes del Proyecto 21.3 (que quizás implemente otro equipo de proyecto).

Proyecto 21.5 Diséñese e impleméntese un sistema basado en la Web para la reserva de aulas en la universidad. Se debe soportar la reserva periódica (días y horas fijas cada semana para un semestre completo). También debe soportarse la cancelación de clases concretas de una reserva periódica.

Puede que también se desee integrarlo con el sistema de matrícula de estudiantes del Proyecto 21.3 (que quizás implemente otro equipo de proyecto) de modo que las aulas puedan reservarse para cursos y las cancelaciones de una clase o de más clases puedan anotarse en una sola interfaz, se reflejen en la reserva de aulas y se comuniquen a los estudiantes por correo electrónico.

Proyecto 21.6 Diséñese e impleméntese un sistema para gestionar exámenes multiopción en línea. Se debe soportar el aporte distribuido de preguntas (por los profesores ayudantes, por ejemplo), la edición de las preguntas por quien esté a cargo del curso y la creación de exámenes a partir del conjunto de preguntas disponible. También se debe poder administrar los exámenes en línea, bien a una hora fija para todos los estudiantes o a cualquier hora, pero con un límite de tiempo desde el comienzo hasta el final (hay que soportar una de las opciones o las dos) y dar a los estudiantes información sobre su puntuación al final del tiempo concedido.

Proyecto 21.7 Diséñese e impleméntese un sistema para gestionar el servicio de correo electrónico de los clientes. El correo entrante va a un buzón común. Hay una serie de agentes de servicio para los clientes que responden el correo electrónico. Si el mensaje es parte de una serie de respuestas (que se siguen

mediante el campo responder a del correo electrónico) es preferible que responda el mensaje el mismo agente que lo respondió anteriormente. El sistema debe realizar un seguimiento de todo el correo entrante y de las respuestas, de modo que los agentes puedan ver el historial de preguntas de cada cliente antes de responder a los mensajes.

Proyecto 21.8 Diséñese e impleméntese un mercado electrónico sencillo en el que los artículos puedan clasificarse para su compra o venta en varias categorías (que deben formar una jerarquía). Puede que también se desee soportar los servicios de alerta, en los que un usuario puede registrar su interés por los artículos de una categoría concreta, quizás con otras restricciones añadidas, sin anunciar su interés de manera pública y se le notifica cuando uno de esos artículos se ofrece a la venta.

Proyecto 21.9 Diséñese e impleméntese un sistema de grupos de noticias basado en la Web. Los usuarios deben poder suscribirse a los grupos de noticias y explorar los artículos de esos grupos. El sistema hace un seguimiento de los artículos que el usuario ha leído, de modo que no vuelvan a exhibirse. También permite la búsqueda de artículos antiguos.

Puede que también se desee ofrecer un servicio de clasificación de los artículos, de modo que los artículos con puntuación elevada se destaquen y se permita al lector ocupado saltarse los artículos con baja puntuación.

Proyecto 21.10 Diséñese e impleméntese un sistema basado en la Web para gestionar una clasificación deportiva. Se registra mucha gente y puede que se les dé alguna clasificación inicial (quizás con base en sus resultados anteriores). Cualquiera puede desafiar a un partido a cualquier otro y las clasificaciones se ajustan de acuerdo con los resultados.

Un sistema sencillo para ajustar las clasificaciones simplemente pasa al ganador por delante del perdedor en la clasificación, en caso de que el ganador estuviera por detrás. Se puede intentar inventar sistemas de ajustes de la clasificación más complicados.

Proyecto 21.11 Diséñese e impleméntese un servicio de listados de publicaciones. El servicio debe permitir la introducción de información sobre las publicaciones, como pueden ser título, autores, año en que apareció la publicación, páginas, etcétera. Los autores deben ser una entidad separada con atributos como nombre, institución, departamento, correo electrónico, dirección y página Web.

La aplicación debe soportar varias vistas de los mismos datos. Por ejemplo, se deben facilitar todas las publicaciones de un autor dado (ordenadas por año, por ejemplo), o todas las publicaciones de los autores de una institución o departamento dado. También se debe soportar la búsqueda por palabras clave, tanto en la base de datos global como en cada una de las vistas.

NOTAS BIBLIOGRÁFICAS

La información sobre los servlets, incluidos los tutoriales, las especificaciones de las normas y el software, está disponible en java.sun.com/products/servlet. La información sobre JSP está disponible en java.sun.com/products/jsp.

Una de las primeras propuestas de pruebas de calidad de sistemas de bases de datos (el índice Wisconsin) la realizaron Bitton et al. [1983]. Los pruebas TPC-A,-B y -C se describen en Gray [1991]. Una versión en línea de todas las descripciones de los pruebas TPC, así como de los resultados de estas pruebas, está disponible en World Wide Web en el URL www.tpc.org; el sitio también contiene información actualizada sobre nuevas propuestas de pruebas. Poess y Floyd [2000] da una introducción de los pruebas TPC-H, TPC-R y TPC-W. El índice OO1 para BDOOs se describe en Cattell y Skeen [1992]; el índice OO7 se describe en Carey et al. [1993].

Kleinrock [1975] y Kleinrock [1976] es un libro de texto popular de dos volúmenes sobre la teoría de colas.

Shasha [1992] ofrece una buena introducción al ajuste de bases de datos. O'Neil y O'Neil [2000] ofrece un tratamiento de libro de texto de muy buena calidad de la medida del rendimiento y de su ajuste. Las reglas de los cinco minutos y del minuto se describen en Gray y Putzolu [1987] y en Gray y Graefe [1997]. Brown et al. [1994] describe una aproximación al ajuste automatizado. La selección de índices y de vistas materializadas se abordan en Ross et al. [1996], Labio et al. [1997], Gupta [1997], Chaudhuri y Narasayya [1997], Agrawal et al. [2000] y Mistry et al. [2001].

La norma nacional americana SQL-86 se describe en ANSI [1986]. La definición de la Arquitectura de aplicaciones de sistemas de IBM de SQL se especifica en IBM [1987]. Las normas para SQL-89 y para SQL-92 están disponibles como ANSI [1989] y como ANSI [1992], respectivamente. Para hallar referencias a la norma SQL:1999 véanse las notas bibliográficas del Capítulo 9.

La interfaz del nivel de llamada X/Open SQL se define en X/Open [1993]; la API ODBC se describe en Microsoft [1997] y en Sanders [1998]. La interfaz X/Open XA se define en X/Open [1991]. Se puede hallar más información sobre ODBC, OLE-DB y ADO en el

sitio Web www.microsoft.com/data, y en varios libros sobre el tema que pueden hallarse mediante www.amazon.com. La norma ODMG 3.0 se define en Cattell [2000]. La revista *Sigmod Record* de *ACM*, que se publica trimestralmente, tiene una sección fija sobre las normas de bases de datos, incluidas las normas de las pruebas de calidad.

Se halla disponible en línea gran cantidad de información sobre las normas basadas en XML. Se puede usar cualquier motor de búsqueda Web como Google para obtener información más detallada y actualizada sobre XML y otras normas.

Loeb [1998] ofrece una detallada descripción de las transacciones electrónicas seguras. La reingeniería de procesos industriales se trata en Cook [1996]. Kirchmer [1999] describe la implementación de aplicaciones mediante software estándar como los paquetes Enterprise Resource Planning (ERP). Umar [1997] trata la reingeniería y aspectos del trabajo con sistemas heredados.

HERRAMIENTAS

Hay muchas herramientas de desarrollo Web que soportan conectividad de bases de datos mediante servlets, JSP, Javascript u otros mecanismos. A continuación se citan algunos de los más conocidos: Java SDK de Sun (java.sun.com), Tomcat (jakarta.apache.org) y servidor Web de Apache (apache.org), WebSphere de IBM (www.software.ibm.com), las herramientas ASP de Microsoft (www.microsoft.com), los productos Coldfusion y JRun de Allaire (www.allaire.com), Resin de Caucho (www.caucho.com) y Zope (www.zope.org). Algunos de ellos, como Apache, son gratis para cualquier tipo de uso y algunos son gratis para uso no comercial o personal, mientras que otros hay que pagarlos. Hay que ver los sitios Web respectivos para obtener más información.

CAPÍTULO

22 CONSULTAS AVANZADAS Y RECUPERACIÓN DE LA INFORMACIÓN

Las empresas han comenzado a aprovechar los cada vez más numerosos datos en línea para tomar mejores decisiones sobre sus actividades, como los artículos que deben tener en inventario y el modo de dirigirse mejor a los clientes para aumentar las ventas. Muchas de las consultas, sin embargo, son bastante complejas y algunos tipos de información no pueden extraerse ni siguiera empleando SQL.

Hay disponibles varias técnicas y herramientas para ayudar en la toma de decisiones. Varias herramientas para el análisis de datos permiten a los analistas ver los datos de diferentes formas. Otras herramientas de análisis realizan un cálculo previo de resúmenes de cantidades muy grandes de datos con objeto de dar respuestas rápidas a las consultas. La norma SQL:1999 contiene ahora estructuras adicionales para soportar el análisis de datos. Otro enfoque para obtener información de los datos es utilizar la *recopilación de datos*, que pretende detectar varios tipos de estructuras en grandes volúmenes de datos. La recopilación de datos complementa varios tipos de técnicas estadísticas con objetivos parecidos.

Los datos de texto también han crecido de manera explosiva. Los datos de texto no están estructurados, a diferencia de los datos rígidamente estructurados de las bases de datos relacionales. La consulta de los datos de texto no estructurados se denomina *recuperación de la información*. Los sistemas de recuperación de la información tienen mucho en común con los sistemas de bases de datos, en particular, el almacenamiento y la recuperación de los datos guardados en almacenamientos secundarios. No obstante, el énfasis en el campo de los sistemas de información es diferente del de los sistemas de bases de datos y se concentra en aspectos como las consultas basadas en palabras clave, la importancia de los documentos para la consulta, y el análisis, la clasificación y el indexado de los documentos.

Este capítulo trata la ayuda a la toma de decisiones, incluidos el procesamiento analítico en línea, la recopilación de datos y la recuperación de la información.

22.1. SISTEMAS DE AYUDA A LA TOMA DE DECISIONES

Las aplicaciones de bases de datos pueden clasificarse grosso modo en procesamiento de transacciones y ayuda a la toma de decisiones, como ya se ha visto en el Apartado 21.3.2. Los sistemas de procesamiento de transacciones se utilizan mucho hoy en día y las empresas han acumulado una enorme cantidad de información generada por esos sistemas.

Por ejemplo, las bases de datos de las empresas suelen contener enormes cantidades de información sobre los clientes y las transacciones. El tamaño del almacenamiento de la información necesario puede llegar a varios centenares de gigabytes o, incluso, a los terabytes, para las cadenas de grandes almacenes. La información de las transacciones de un gran almacén puede incluir el nombre o identificador (como puede ser el número de la tarjeta de crédito) del cliente, los artículos adquiridos, el precio pagado y las fechas en que se realizaron las compras. La información sobre los artículos adquiridos puede incluir el nombre del artículo, el fabricante, el número del modelo, el color y la talla. La información sobre los clientes puede incluir su historial de crédito, sus ingresos anuales, su domicilio, su edad e, incluso, su nivel académico.

Estas bases de datos de gran tamaño pueden resultar minas de información para adoptar decisiones empresariales, como los artículos que debe haber en inventario y los descuentos que hay que ofrecer. Por ejemplo, puede que una cadena de grandes almacenes note un aumento súbito de las compras de camisas de franela en la Sierra de Guadarrama, darse cuenta de que hay una tendencia y comenzar a almacenar un mayor número de esas camisas en las tiendas de esa zona. O puede que una empresa automovilística descubra, al consultar su base de datos, que la mayor parte de los coches deportivos de pequeño tamaño los compran mujeres jóvenes cuyos ingresos anuales superan los 50.000 €. Puede que la empresa dirija su publicidad para que atraiga más mujeres de esas características a que compren coches deportivos de pequeño tamaño y evite desperdiciar dinero intentando atraer a otras categorías de consumidores

para que compren esos coches. En ambos casos la empresa ha identificado pautas de comportamiento de los consumidores y las ha utilizado para adoptar decisiones empresariales.

El almacenamiento y recuperación de los datos para la ayuda a la toma de decisiones plantea varios problemas:

- Aunque muchas consultas para ayuda a la toma de decisiones pueden escribirse en SQL, otras no pueden expresarse en SQL o no pueden hacerlo con facilidad. En consecuencia, se han propuesto varias extensiones de SQL para facilitar el análisis de los datos. El área de *procesamiento analítico en línea* (*Online Analytical Processing,* OLAP) trata de las herramientas y de las técnicas para el análisis de los datos que pueden dar respuestas casi instantáneas a las consultas que soliciten datos resumidos, aunque la base de datos sea extremadamente grande. En el Apartado 22.2 se estudian las extensiones de SQL para el análisis de datos y las técnicas para el procesamiento analítico en línea.
- Los lenguajes de consulta de bases de datos no resultan adecuados para el rendimiento de los **análisis estadísticos** detallado de los datos. Hay varios paquetes, como SAS y S++, que ayudan en el análisis estadístico. A estos paquetes se les han añadido interfaces con las bases de datos para permitir que se almacenen en la base de datos grandes volúmenes de datos y se recuperen de manera eficiente para su análisis. El campo del análisis estadístico es una gran disciplina por sí misma, véanse las referencias en las notas bibliográficas para obtener más información.
- Las técnicas de búsqueda de información intentan descubrir de manera automática las reglas y las pautas estadísticas de los datos. El campo de la *recopilación de datos* combina las técnicas de búsqueda de información creadas por los investigadores en inteligencia artificial y los expertos en análisis estadístico con las técnicas de implementación eficiente que permiten utilizarlas en bases de datos extremadamente grandes. El Apartado 22.3 estudia la recopilación de datos.
- Las grandes empresas tienen varios orígenes de datos que necesitan utilizar para adoptar decisiones empresariales. Los orígenes pueden almacenar los datos según diferentes esquemas. Por motivos de rendimiento (así como por motivos de control de la organización) los orígenes de datos no suelen permitir que otras partes de la empresa recuperen datos a petición.

  Para ejecutar de manera eficiente las consultas sobre datos tan diferentes las empresas han creado *almacenes de datos*. Los almacenes de datos reúnen los datos de varios orígenes bajo un esquema unificado en un solo sitio. Por tanto, ofrecen al usuario una sola interfaz uniforme para los datos. Los problemas de la creación y mantenimiento de los almacenes de datos se estudian en el Apartado 22.4.

El área de **ayuda a la toma de decisiones** puede considerarse que abarca todas las áreas anteriores, aunque algunas personas utilizan el término en un sentido más restrictivo que excluye el análisis estadístico y la recopilación de datos.

22.2. ANÁLISIS DE DATOS Y OLAP

Aunque es mejor dejar el análisis estadístico complejo a los paquetes estadísticos, las bases de datos deben soportar las formas sencillas, utilizadas frecuentemente, de análisis estadístico. Dado que los datos almacenados en las bases de datos suelen ser de gran volumen, hay que resumirlos de algún modo si hay que obtener información que puedan utilizar los usuarios.

Las herramientas OLAP soportan el análisis interactivo de la información de resumen. Se han desarrollado varias extensiones de SQL para soportar las herramientas OLAP. Hay muchas tareas utilizadas con frecuencia que no pueden realizarse empleando las facilidades básicas de agregación y agrupamiento de SQL. Entre los ejemplos se hallan la búsqueda de percentiles, las distribuciones acumulativas o los agregados sobre ventanas deslizantes de datos ordenados secuencialmente. Recientemente se han propuesto varias extensiones de SQL para soportar estas tareas y se han implementado en productos como Oracle y DB2 de IBM.

22.2.1. Procesamiento analítico en línea

El análisis estadístico suele necesitar el agrupamiento de varios atributos. Considérese una aplicación en que una tienda desea averiguar las prendas que son más populares. Supóngase que las prendas están caracterizadas por su nombre de artículo, su color y su talla y que se tiene la relación *ventas* con el esquema *ventas* (*nombre-artículo, color, talla, número*). Supóngase que *nombre-artículo* puede adoptar los valores (falda, vestido, camisa, pantalón), *color* puede adoptar los valores (oscuro, pastel, blanco) y *talla* puede adoptar los valores (pequeña, mediana, grande).

Dada una relación utilizada para el análisis de datos se pueden identificar algunos de sus atributos como **atributos de medida**, ya que miden algún valor y pueden agregarse. Por ejemplo, el atributo *número* de la relación *ventas* es un atributo de medida, ya que mide la cantidad de unidades vendidas. Algunos de los demás atributos (o todos ellos) de la relación se identifican como

atributos de dimensión, ya que definen las dimensiones en las que se ven los atributos de medida y los resúmenes de los atributos de medida. En la relación *ventas*, *nombre-artículo, color* y *talla* son atributos de dimensión. (Una versión más realista de la relación *ventas* tendría más dimensiones, como tiempo o lugar de venta, y más medidas como el valor monetario de la venta.)

Los datos que pueden modelarse como atributos de dimensión y como atributos de medida se denominan **datos multidimensionales**.

Para analizar los datos multidimensionales puede que el administrador desee ver los datos dispuestos como se ven en la tabla de la Figura 22.1. La tabla muestra las cifras totales de diferentes combinaciones de *nombre-artículo* y *color*. El valor de *talla* se especifica como **todas**, lo que indica que los valores mostrados son un resumen para todos los valores de *talla*.

La tabla de la Figura 22.1 es un ejemplo de **tabulación cruzada**, también denominada **tabla dinámica**. En general, las tabulaciones cruzadas son tablas en las que los valores de uno de los atributos (por ejemplo, *A*) forman las cabeceras de las filas, los valores del otro atributo (por ejemplo, *B*) forman las cabeceras de las columnas y los valores de cada celda se obtienen como sigue: cada celda puede identificarse como (a_i, b_j), donde a_i es un valor de *A* y b_j un valor de *B*. Si hay como máximo una tupla con cualquier valor de (a_i, b_j), el valor de la celda se obtiene de esa única tupla (si es que hay alguna); por ejemplo, puede ser el valor de uno o varios atributos de la tupla. Si puede haber varias tuplas con el valor (a_i, b_j), el valor de la celda debe obtenerse por agregación de las tuplas con ese valor. En este ejemplo la agregación utilizada es la suma de los valores del atributo *número*. En este ejemplo la tabulación cruzada también tiene una columna y una fila adicionales que guardan los totales de las celdas de cada fila o columna. La mayor parte de las tabulaciones cruzadas tienen esas filas y columnas de resumen.

Las tabulaciones cruzadas son diferentes de las tablas relacionales que suelen guardarse en las bases de datos, ya que el número de columnas de la tabulación cruzada depende de los datos. Una modificación en los valores de los datos puede dar lugar a que se añadan más columnas, lo que no resulta deseable para el almacenamiento de los datos. No obstante, la vista de tabulación cruzada es deseable para mostrársela a los usuarios. La representación de las tabulaciones cruzadas sin valores resumen en un formulario relacional con un número fijo de columnas es directa. La tabulación cruzada con columnas o filas resumen puede representarse introduciendo el valor especial **all** (todos) para representar los subtotales, como en la Figura 22.2. La norma SQL:1999 utiliza realmente el valor **null** (nulo) en lugar de **all**; pero, para evitar confusiones con los valores nulos habituales, en el libro se seguirá utilizando **all**.

Considérense las tuplas (falda, **all**, 53) y (vestido, **all**, 35). Se han obtenido estas tuplas eliminando las tuplas individuales con diferentes valores de *color* y sustituyendo el valor de *número* por un agregado, es decir, una suma. El valor **all** puede considerarse representante del conjunto de los valores de un atributo. Las tuplas con el valor **all** sólo para la dimensión *color* pueden obtenerse mediante una consulta de SQL que lleve a cabo una agrupación en la columna *nombre-artículo*. De manera parecida, se puede utilizar una agrupación en *color* para conseguir las tuplas con el valor **all** para

nombre-artículo	*color*	*número*
falda	oscuro	8
falda	pastel	35
falda	blanco	10
falda	**all**	53
vestido	oscuro	20
vestido	pastel	10
vestido	blanco	5
vestido	**all**	35
camisa	oscuro	14
camisa	pastel	7
camisa	blanco	28
camisa	**all**	49
pantalones	oscuro	20
pantalones	pastel	2
pantalones	blanco	5
pantalones	**all**	27
all	oscuro	62
all	pastel	54
all	blanco	48
all	**all**	164

FIGURA 22.2. Representación relacional de los datos de la Figura 22.1.

talla: **all**

nombre-artículo \ *color*	oscuro	pastel	blanco	Total
falda	8	35	10	53
vestido	20	10	5	35
camisa	14	7	28	49
pantalón	20	2	5	27
Total	62	54	48	164

FIGURA 22.1. Tabulación cruzada de *ventas* con *nombre-artículo* y *color*.

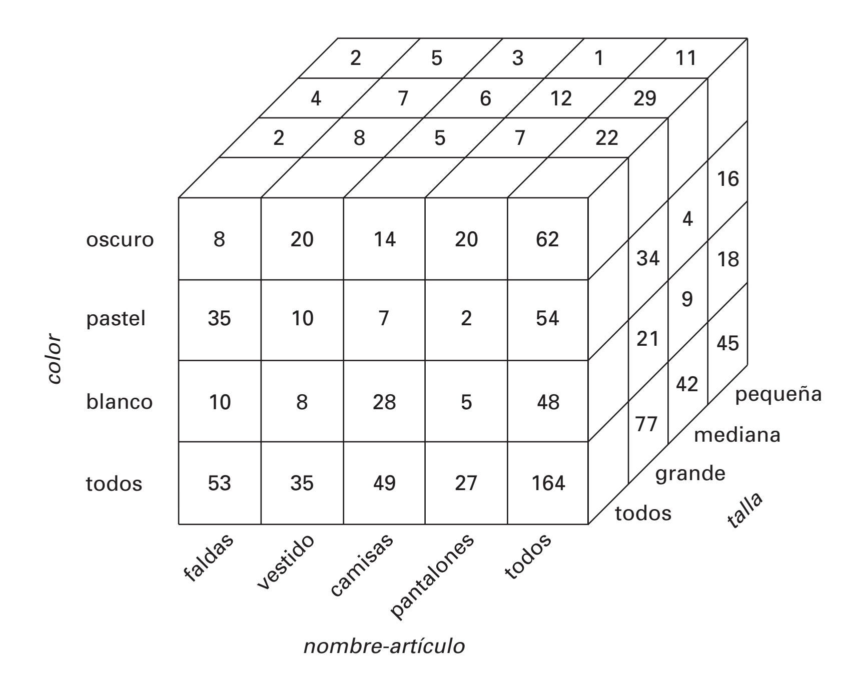


FIGURA 22.3. Cubo de datos tridimensional.

nombre-artículo, y una agrupación sin atributo alguno (que en SQL puede omitirse simplemente) puede utilizarse para obtener la tupla con el valor **all** para *nombre-artículo* y *color*.

La generalización de las tabulaciones cruzadas, que son bidimensionales, a *n* dimensiones puede visualizarse como cubos *n* dimensionales, denominados **cubos de datos**. La Figura 22.3 muestra un cubo de datos para la relación *ventas*. El cubo de datos tiene tres dimensiones, a saber, *nombre-artículo*, *color* y *talla*, y el atributo de medida es *número*. Cada celda se identifica por los valores de estas tres dimensiones. Cada celda del cubo de datos contiene un valor, igual que en la tabulación cruzada. En la Figura 22.3, el valor contenido en la celda se muestra en una de las caras de la celda; las otras caras de la celda se muestran en blanco si son visibles.

El valor de una dimensión puede ser **all**, en cuyo caso la celda contiene un resumen de todos los valores de esa dimensión, como en el caso de las tabulaciones cruzadas. El número de maneras diferentes en que las tuplas pueden agruparse para su agregación puede ser grande. De hecho, para una tabla con *n* dimensiones, se puede realizar la agregación con la agrupación de cada uno de los 2^n subconjuntos de las *n* dimensiones[1].

El sistema de procesamiento analítico en línea o sistema OLAP es un sistema interactivo que permite a los analistas ver diferentes resúmenes de los datos multidimensionales. Las palabras *en línea* indican que los analistas deben poder solicitar nuevos resúmenes y obtener respuestas en línea, en pocos segundos, y no deberían verse obligados a esperar mucho tiempo para ver el resultado de las consultas.

Con los sistemas OLAP los analistas de datos pueden ver diferentes tabulaciones cruzadas para los mismos datos seleccionando de manera interactiva los atributos de las tabulaciones cruzadas. Cada tabulación cruzada es una vista bidimensional del cubo de datos multidimensional. Por ejemplo, el analista puede seleccionar una tabulación cruzada para *nombre-artículo* y *talla* o una tabulación cruzada para *color* y *talla*. La operación de modificación de las dimensiones utilizadas en las tabulaciones cruzadas se denomina **pivotaje**.

Los sistemas OLAP también ofrecen otras funcionalidades. Por ejemplo, puede que un analista desee ver una tabulación cruzada de *nombre-artículo* y *color* para un valor fijo de *talla*, por ejemplo, grande, en lugar de la suma para todas las tallas. Esta operación se conoce como **corte**, ya que puede considerarse como la vista de una rebanada del cubo de datos. A veces la operación se denomina **corte de cubos**, especialmente cuando se fijan los valores de varias dimensiones.

Cuando se utilizan tabulaciones cruzadas para ver cubos multidimensionales los valores de los atributos de dimensión que no forman parte de la tabulación cruzada se muestran por encima de ella. El valor de estos atributos puede ser **all**, como puede verse en la Figura 22.1, lo que indica que los datos de la tabulación cruzada son un resumen de todos los valores del atributo. El corte y la creación de datos consisten simplemente

[1] La agrupación sobre el conjunto de las *n* dimensiones sólo resulta útil si la tabla puede tener duplicados.

en la selección de valores concretos de estos atributos, que se muestran luego por encima de la tabulación cruzada.

Los sistemas OLAP permiten a los usuarios ver los datos con cualquier nivel de granularidad que se desee. La operación de pasar de datos con una granularidad más fina a una granularidad más gruesa (mediante la agregación) se denomina **abstracción**. En este ejemplo, a partir del cubo de datos para la tabla *ventas*, se obtiene la tabulación cruzada de ejemplo abstrayendo el atributo *talla*. La operación inversa —la de pasar de datos con una granularidad más gruesa a una más fina— se denomina **concreción**. Claramente, los datos con granularidad más fina no pueden generarse a partir de datos con una granularidad más gruesa; deben generarse a partir de los datos originales o de datos resumidos de granularidad aún más fina.

Puede que un analista desee ver una dimensión con niveles diferentes de detalle. Por ejemplo, los atributos de tipo **FechaHora** contienen una fecha y una hora del día. El empleo de horas con precisión de segundos (o menos) puede que no sea significativo: los analistas que estén interesados en la hora aproximada del día puede que sólo miren el valor horario. Los analistas que estén interesados en las ventas de cada día de la semana puede que apliquen la fecha al día de la semana y sólo se fijen en eso. Puede que otro analista esté interesado en agregados mensuales, trimestrales o de años enteros.

Los diferentes niveles de detalle de los atributos pueden organizarse en una **jerarquía**. La Figura 22.4(a) muestra una jerarquía para el atributo **FechaHora**. La Figura 22.4(b), que puede ser otro ejemplo, muestra una jerarquía para la ubicación, con la ciudad en la parte inferior de la jerarquía, el estado por encima, el país en el nivel siguiente y la región en el nivel superior. En el ejemplo anterior la ropa puede agruparse por categorías (por ejemplo, ropa de hombre o de mujer); la *categoría* estaría por encima de *nombre-artículo* en la jerarquía de la ropa. En el nivel de los valores reales las faldas y los vestidos caerían dentro de la categoría de ropa de mujer y los pantalones y las camisas en la de ropa de hombre.

Puede que un analista esté interesado en ver las ventas de ropa divididas entre ropa de hombre y ropa de mujer y que no esté interesado en ver los valores individuales. Tras ver los agregados en el nivel de ropa de hombre y ropa de mujer puede que el analista *concrete la jerarquía* para ver los valores individuales. Un analista que examine el nivel detallado puede *abstraer la jerarquía* y examinar agregados de niveles más gruesos. Ambos niveles pueden mostrarse en la misma tabulación cruzada, como en la Figura 22.5.

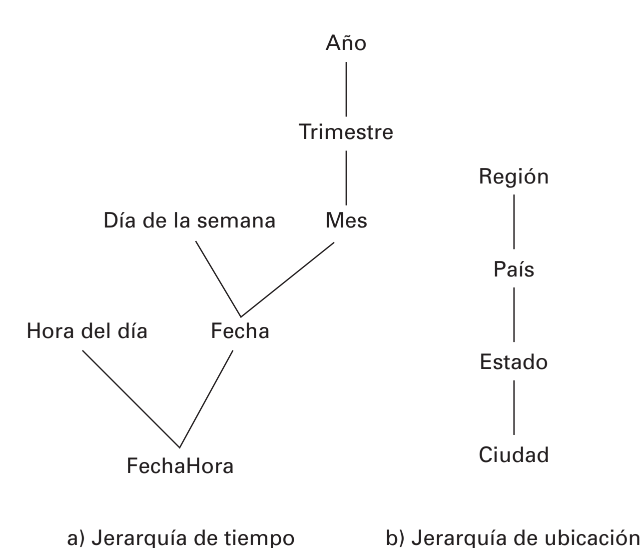


FIGURA 22.4. Jerarquías de las dimensiones.

22.2.2. Implementación de OLAP

Los primeros sistemas de OLAP utilizaban arrays de memoria multidimensionales para almacenar los cubos de datos y se denominaban sistemas **OLAP multidimensionales (Multidimensional OLAP, MOLAP)**. Posteriormente, los servicios OLAP se integraron en los sistemas relacionales y los datos se almacenaron en las bases de datos relacionales. Estos sistemas se denominan sistemas **OLAP relacionales (Relational OLAP, ROLAP)**. Los sistemas híbridos, que almacenan algunos resúmenes en la memoria y los datos básicos y otros resúmenes en bases de datos relacionales, se denominan sistemas **OLAP híbridos (Hybrid OLAP, HOLAP)**.

Muchos sistemas OLAP se implementan como sistemas cliente-servidor. El servidor contiene la base de datos relacional y los cubos de datos MOLAP. Los sistemas clientes obtienen vistas de los datos comunicándose con el servidor.

Una manera ingenua de calcular todo el cubo de datos (todas las agrupaciones) de una relación es utilizar cualquier algoritmo estándar para calcular las operaciones de agregación, agrupación a agrupación. El algoritmo ingenuo necesita un gran número de exploraciones de la relación. Una optimización sencilla consiste en calcular el agregado para, por ejemplo (*nombre-artículo*, *color*) a partir del agregado (*nombre-artículo*, *color*, *talla*), en lugar de hacerlo a partir de la relación original. Para las funciones de agregación estándar de SQL se pueden calcular agregados con agrupaciones sobre un conjunto de atributos A a partir de un agregado con agrupación sobre un conjunto de atributos B si $A \subseteq B$; se puede hacer como ejercicio (véase el Ejercicio 22.1), pero hay que tener en cuenta que al calcular **avg** también hace falta el valor **count**. (Para algunas funciones de agregación no estándar como la mediana, los agregados no pueden calcularse de la manera indicada; la optimización aquí descrita no es aplicable a estas funciones de agregación «*no-descomponibles*».) La cantidad de datos que se lee disminuye de manera significativa al calcular los agregados a partir de otros agregados, en lugar de hacerlo a partir de la relación original. Se pueden conseguir otras mejoras; por ejemplo, se pue-

categoría	*nombre-artículo*					
		oscura	pastel	blanca	total	
ropa de hombre	camisa	8	8	10	53	
	pantalones	20	20	5	35	
	subtotal	28	28	15		88
ropa de mujer	falda	14	14	28	49	
	vestido	20	20	5	27	
	subtotal	34	34	33		76
total		62	62	48		164

FIGURA 22.5. Tabulación cruzada de *ventas* con la jerarquía para *nombre-artículo*.

den calcular varias agrupaciones con una sola lectura de los datos. Véanse las notas bibliográficas para hallar referencias a los algoritmos para el cálculo eficiente de cubos de datos.

Las primeras implementaciones OLAP calculaban previamente los cubos de datos completos, es decir, las agrupaciones por todos los subconjuntos de los atributos de dimensión, y los almacenaban. El cálculo previo permite que las consultas OLAP se respondan en pocos segundos, incluso para conjuntos de datos que pueden contener millones de tuplas que suponen gigabytes de datos. No obstante, hay 2^n agrupaciones con n atributos de dimensión; las jerarquías de los atributos aumentan más aún el número. En consecuencia, todo el cubo de datos suele ser mayor que la relación original que lo generó y, en muchos casos, no resulta posible almacenarlo entero.

En lugar de calcular previamente todas las agrupaciones posibles y almacenarlas, resulta razonable calcular previamente algunas de las agrupaciones y almacenarlas y calcular el resto según se soliciten. En lugar de calcular las consultas a partir de la relación original, lo que puede tardar mucho tiempo, se pueden calcular a partir de otras consultas calculadas previamente. Por ejemplo, supóngase que una consulta necesita resúmenes según (*nombre-artículo*, *color*), que no se ha calculado con anterioridad. El resultado de la consulta puede calcularse a partir de resúmenes según (*nombre-artículo*, *color*, *talla*), si ya se ha calculado. Véanse las notas bibliográficas para hallar referencias al modo de seleccionar para su cálculo previo un buen conjunto de agrupaciones, dados los límites de almacenamiento disponible para los resultados calculados previamente.

Los datos de los cubos no se pueden generar mediante una sola consulta SQL utilizando las estructuras básicas **group by**, ya que los agregados se calculan para varias agrupaciones diferentes de los atributos de dimensión. El Apartado 22.2.3 estudia las extensiones de SQL para el soporte de la funcionalidad OLAP.

22.2.3. Ampliación de la agregación

La funcionalidad de agregación de SQL-92 está limitada, por lo que diferentes bases de datos han implementado varias extensiones. No obstante, la norma SQL:1999 define un amplio conjunto de funciones de agregación, que se describen en este apartado y en los dos siguientes. Las bases de datos de Oracle y de DB2 de IBM soportan la mayor parte de estas características y, sin duda, otras bases de datos soportarán estas características en un futuro próximo.

Las nuevas funciones de agregación para un solo atributo son la desviación estándar y la varianza (**stddev** y **variance**). La desviación estándar es la raíz cuadrada de la varianza[2]. Algunos sistemas de bases de datos soportan otras funciones de agregación como la mediana y la moda. Algunos sistemas de bases de datos incluso permiten que los usuarios añadan nuevas funciones de agregación.

SQL:1999 también soporta una nueva clase de **funciones de agregación binarias**, que pueden calcular resultados estadísticos para parejas de atributos; entre ellas están las correlaciones, las covarianzas y las curvas de regresión, que dan una línea que aproxima la relación entre los valores de la pareja de atributos. Las definiciones de estas funciones pueden hallarse en cualquier libro de texto estándar, como los que se citan en las notas bibliográficas.

SQL:1999 también soporta generalizaciones de la estructura **group by**, mediante las estructuras **cube** y **rollup**. Un uso representativo de la estructura **cube** es el siguiente:

select *nombre-artículo, color, talla*, **sum**(*número*)
from *ventas*
group by cube(*nombre-artículo, color, talla*)

Esta consulta calcula la unión de ocho agrupaciones diferentes de la relación *ventas*:

[2] La norma SQL:1999 soporta en realidad dos tipos de varianza, denominadas varianza de la población y varianza de la muestra y, por tanto, dos tipos de desviación estándar. La definición de los dos tipos difiere ligeramente; los detalles se pueden consultar en cualquier libro de texto de estadística.

{ (*nombre-artículo, color, talla*), (*nombre-artículo, color*), (*nombre-artículo, talla*), (*color, talla*), (*nombre-artículo*), (*color*), (*talla*), () }

donde () denota una lista **group by** vacía.

Para cada agrupación el resultado contiene el valor nulo para los atributos no presentes en la agrupación. Por ejemplo, la tabla de la Figura 22.2, sustituyendo **all** por *null*, la puede calcular la consulta

select *nombre-artículo, color*, **sum**(*número*)
from *ventas*
group by cube(*nombre-artículo, color*)

Una estructura **rollup** representativa es

selectC *nombre-artículo, color, talla*, **sum**(*número*)
from *ventas*
group by rollup(*nombre-artículo, color, talla*)

En este caso sólo se han generado cuatro agrupaciones:

{ (*nombre-artículo, color, talla*), (*nombre-artículo, color*), (*nombre-artículo*), () }

La instrucción **rollup** puede utilizarse para generar agregados en varios niveles de una jerarquía para una columna. Por ejemplo, supóngase que se tiene la tabla *categoríaartículo*(*nombre-artículo, categoría*) que da la categoría de cada artículo. La consulta

select *categoría, nombre-artículo*, **sum**(*número*)
from *ventas*, *categoríaartículo*
where *ventas.nombre-artículo*
= *categoríaartículo.nombre-artículo*
group by rollup(*categoría, nombre-artículo*)

da un resumen jerárquico según *nombre-artículo* y según *categoría*.

Se pueden utilizar varios **rollup** y varios **cube** en una sola cláusula group by. Por ejemplo, la consulta siguiente

select *nombre-artículo, color, talla*, **sum**(*número*)
from *ventas*
group by rollup(*nombre-artículo*), **rollup**(*color, talla*)

genera las agrupaciones

{ (*nombre-artículo, color, talla*), (*nombre-artículo, color*), (*nombre-artículo*), (*color, talla*), (*color*), () }

Para comprender el motivo hay que tener en cuenta que **rollup**(*nombre-artículo*) genera dos agrupaciones, {(*nombreartículo*), ()}, y que **rollup**(*color, talla*) genera tres agrupaciones, {(*color, talla*), (*color*), () }. El producto cartesiano de los dos da como resultado las seis agrupaciones mostradas.

Como ya se ha mencionado en el Apartado 22.2.1, SQL:1999 utiliza el valor **null** para indicar el sentido habitual de nulo así como **all**. Este uso dual de **null** puede generar ambigüedad si los atributos utilizados en una cláusula rollup o en una cláusula cube contienen valores nulos. Se puede aplicar la función **grouping** a un atributo; devuelve 1 si el valor es un valor nulo que represente a **all**, y devuelve 0 en los demás casos. Considérese la consulta siguiente:

select *nombre-artículo, color, talla*, **sum**(*número*),
grouping(*nombre-artículo*)
as *indicador-nombre-artículo*,
grouping(*color*) **as** *indicador-color*,
grouping(*talla*) **as** *indicador-talla*
from *ventas*
group by cube(*nombre-artículo, color, talla*)

El resultado es el mismo que en la versión de la consulta sin **grouping**, pero con tres columnas adicionales denominadas *indicador-nombre-artículo, indicador-color* y *indicador-talla*. En cada tupla el valor de los campos indicador es 1 si el campo correspondiente es un valor nulo que respresente a **all**.

En lugar de utilizar etiquetas para indicar los valores nulos que representan a **all**, se pueden sustituir los valores nulos por un valor a la elección del usuario:

decode(**grouping**(*nombre-artículo*), 1, 'todos', *nombre-artículo*)

Esta expresión devuelve el valor «todos» si el valor de *nombre-artículo* es un valor nulo que se corresponda con **all**, y devuelve el valor real de *nombre-artículo* en caso contrario. Esta expresión puede utilizarse en lugar de *nombre-artículo* en la cláusula select para obtener «todos» en el resultado de la consulta, en lugar de valores nulos que representen a **all**.

Ni la cláusula **rollup** ni la cláusula **cube** ofrecen un control completo de las agrupaciones que se generan. Por ejemplo, no se pueden utilizar para especificar que sólo se desean las agrupaciones {(*color, talla*), (*talla, nombre-artículo*)}. Estas agrupaciones restringidas pueden generarse utilizando la estructura **grouping** en la cláusula **having**; los detalles se dejan como ejercicio para el lector.

22.2.4. Clasificación

Hallar la posición de un valor en un conjunto más grande es una operación frecuente. Por ejemplo, puede que se desee asignar una clasificación a los estudiantes de acuerdo con sus notas totales, con el puesto 1 para el estudiante con las notas más altas, el puesto 2 para el estudiante con las segundas mejores notas, etcétera. Aunque estas consultas pueden expresarse en SQL-92, resultan difíciles de expresar e ineficientes a la hora de la evaluación. Los programadores suelen recurrir a escri-

bir en parte la consulta en SQL y en parte en un lenguaje de programación. Un tipo de consulta relacionado es la búsqueda del percentil que le corresponde a un valor de un (multi)conjunto, por ejemplo, el tercio inferior, el tercio central o el tercio superior. Aquí se estudiará el soporte de SQL:1999 para estos tipos de consultas.

La clasificación se realiza conjuntamente con una especificación **order by**. Supóngase que se tiene una relación *estudiante-notas*(*id-estudiante, notas*) que almacena las notas obtenidas por cada estudiante. La consulta siguiente da la clasificación de cada estudiante.

select *id-estudiante*, **rank**() **over** (**order by** (*notas*)
  desc) **as** *clasificación-e*
from *notas-estudiante*

Obsérvese que el orden de las tuplas en el resultado no se ha definido, por lo que puede que no estén ordenadas según su clasificación. Se necesita una cláusula **order by** adicional para dejarlas ordenadas, como puede verse a continuación.

select *id-estudiante*, **rank** () **over** (**order by** (*notas*)
  desc) **as** *clasificación-e*
from *notas-estudiante* **order by** *clasificación-e*

Un aspecto básico de las clasificaciones es el modo de tratar el caso de que haya varias tuplas que sean iguales en el atributo o atributos de ordenación. En el ejemplo anterior esto significa decidir lo que se hace si hay dos estudiantes con la mismas notas. La función **rank** da la misma clasificación a todas las tuplas que sean iguales en los atributos **order by**. Por ejemplo, si la nota más elevada la comparten dos estudiantes, los dos obtendrían el puesto 1. El puesto siguiente que se diera sería el 3, no el 2, por lo que si tres estudiantes consiguieran la siguiente nota más alta, todos ellos obtendrían el puesto 3, y los siguientes estudiantes obtendrían el puesto 5, etcétera. También hay una función **dense rank** que no crea saltos en la ordenación. En el ejemplo anterior las tuplas con el segundo valor más alto obtienen el puesto 2 y las tuplas con el tercer valor más elevado obtienen el puesto 3, etcétera.

Se puede llevar a cabo la clasificación dentro de particiones de los datos. Por ejemplo, supóngase que se tiene una relación adicional *estudiante-sección*(*id-estudiante, sección*) que almacena para cada estudiante la sección en la que estudia. La consulta siguiente da la clasificación de los estudiantes dentro de cada sección.

select *id-estudiante*, *sección*,
  rank () **over** (**partition by** *sección*
  order by *notas* **desc**) **as** *clasificación-sec*
from *notas-estudiante*, *sección-estudiante*
where *notas-estudiante.id-estudiante*
  = *sección-estudiante.id-estudiante*
order by *sección*, *clasificación-sec*

La cláusula **order by** externa ordena las tuplas del resultado por secciones y, dentro de cada sección, por su clasificación.

Se pueden utilizar varias expresiones **rank** dentro de cada sentencia select; por tanto, se puede obtener la clasificación general y la clasificación dentro de cada sección empleando dos expresiones **rank** en la misma cláusula **select**. Una pregunta interesante es lo que ocurre cuando se produce una clasificación (posiblemente con partición) junto con una cláusula **group by**. En ese caso, la cláusula **group by** se aplica en primer lugar y la partición y la clasificación se realizan sobre los resultados de la cláusula group by. Así, los valores agregados pueden utilizarse para la clasificación. Por ejemplo, supóngase que se tienen las notas de cada estudiante en varias asignaturas. Para clasificar a los estudiantes por la suma de sus notas en varias asignaturas se puede utilizar una cláusula **group by** para calcular las notas agregadas de cada estudiante y luego clasificar a los estudiantes por la suma agregada. Los detalles se le dejan al lector como ejercicio.

Las funciones de clasificación pueden utilizarse para hallar las *n* primeras tuplas incrustando una consulta de clasificación en una consulta de un nivel exterior; los detalles se dejan para un ejercicio. Téngase en cuenta que las *n* últimas tuplas son simplemente las mismas que las *n* primeras con un orden inverso. Varios sistemas de bases de datos ofrecen extensiones no estándar de SQL para especificar directamente que sólo se necesitan los *n* primeros resultados; esas extensiones no necesitan la función de clasificación y simplifican el trabajo del optimizador, pero (actualmente) no son tan generales, ya que no soportan las particiones.

SQL:1999 también especifica otras funciones que pueden utilizarse en lugar de **rank**. Por ejemplo, **percent rank** de una tupla da la clasificación de la tupla en forma de fracción. Si hay *n* tuplas en la partición[3] y la clasificación de la tupla es *r*, su clasificación percentual se define como $(r - 1)/(n - 1)$ (y como nula si sólo hay una tupla en la partición). La función **cume_dist**, abreviatura de distribución acumulativa (cumulative distribution), para una tupla se define como p/n, donde *p* es el número de tuplas de la partición con valores de ordenación que preceden o son iguales al valor de ordenación de la tupla, y *n* es el número de tuplas de la partición. La función **row number** ordena las filas y da a cada una un número único correspondiente a su posición en el orden; filas diferentes con el mismo valor de ordenación reciben números de fila diferentes, de manera no determinista.

[3] Todo el conjunto se trata como una sola partición si no se utiliza ninguna partición explícita.

Finalmente, para una constante dada *n*, la función de clasificación **ntile**(*n*) toma las tuplas de cada partición en el orden especificado y las divide en *n* cajones con igual número de tuplas[4]. Para cada tupla, **ntile**(*n*) da el número del cajón en el que se halla, con los números de los cajones comenzando por 1. Esta función resulta especialmente útil para la creación de histogramas basados en percentiles. Por ejemplo, se pueden ordenar los empleados por su salario y utilizar **ntile**(3) para hallar el rango (tercio inferior, tercio central o tercio superior) en el que se halla cada empleado, y para calcular el salario total ganado por los empleados de cada rango:

```
select tercil, sum(sueldo)
from (
      select sueldo, ntile(3) over (order by (sueldo))
            as tercil
      from empleado) as s
group by tercil.
```

La presencia de valores nulos puede complicar la definición de la clasificación, dado que no está claro si deben colocarse antes en el orden. SQL:1999 permite que el usuario especifique dónde deben aparecer mediante **nulls first** o **nulls last**, por ejemplo

```
select id-estudiante, rank () over (order by notas
      desc nulls last) as clasificación-e
from notas-estudiante
```

22.2.5. Ventanas

Un ejemplo de consulta *ventana* es una consulta que, dados los valores de ventas para cada fecha, calcula para cada fecha el promedio de ventas de ese día, del día anterior y del día siguiente; esas consultas de media móvil se utilizan para suavizar las variaciones aleatorias. Otro ejemplo de consulta ventana es la consulta que halla el saldo acumulado de una cuenta, dada una relación que especifique las imposiciones y las retiradas de fondos de la cuenta. Esas consultas resultan difíciles o imposibles de expresar (depende de la consulta en concreto) en SQL básico.

SQL:1999 ofrece una característica de ventanas para soportar esas consultas. A diferencia de **group by**, la misma tupla puede estar en varias ventanas. Supóngase que se tiene la relación *transacción*(*número-cuenta, fecha-hora, valor*), donde *valor* es positivo para las imposiciones de fondos y negativo para las retiradas. Se da por supuesto que hay como máximo una transacción por cada valor *fecha-hora*.

Considérese la consulta

```
select número-cuenta, fecha-hora,
      sum(valor) over
            (partition by número-cuenta
             order by fecha-hora
             rows unbounded preceding)
      as saldo
from transacción
order by número-cuenta, fecha-hora
```

La consulta da los saldos acumulados de cada cuenta justo antes de cada transacción en esa cuenta; el saldo acumulado de una cuenta es la suma de valores de todas las transacciones anteriores de la cuenta.

La cláusula **partition by** separa las tuplas por número de cuenta, de modo que para cada fila sólo se consideren las tuplas de su partición. Se crea una ventana para cada tupla; las palabras clave **rows unbounded preceding** especifican que la ventana de cada tupla consiste en todas las tuplas de la partición que la preceden en el orden especificado (en este caso, orden creciente de *fecha-hora*). La función de agregación **sum**(*valor*) se aplica a todas las tuplas de la ventana. Obsérvese que la consulta no utiliza ninguna cláusula **group by**, ya que hay una tupla de resultado por cada tupla de la relación *transacción*.

Aunque la consulta puede escribirse sin las estructuras ampliadas, sería bastante difícil de formular. Obsérvese también que se pueden superponer diferentes ventanas, es decir, una tupla puede estar presente en más de una ventana.

Se pueden especificar otros tipos de ventanas. Por ejemplo, para obtener una ventana que contenga las 10 filas anteriores a cada fila, se puede especificar **rows** 10 **preceding**. Para obtener una ventana que contenga la fila actual, la anterior y la siguiente se puede utilizar **between rows** 1 **preceding and** 1 **following**. Para obtener las filas siguientes y la fila actual se puede decir **between rows unbounded preceding and current**. Obsérvese que si la ordenación se realiza sobre un atributo que no sea clave el resultado no es determinista, ya que el orden de las tuplas no está definido completamente.

Se pueden especificar ventanas mediante rangos de valores, en lugar de hacerlo mediante número de filas. Por ejemplo, supóngase que el valor de ordenación de una tupla es *v*; entonces, **range between** 10 **preceding and current row** dará tuplas cuyo valor de ordenación se halle entre $v - 10$ y *v* (ambos valores incluidos). Al tratar con fechas se puede utilizar **range interval** 10 **day preceding** para obtener una ventana que contenga tuplas con los 10 días anteriores, pero sin incluir la fecha de la tupla.

Evidentemente, la funcionalidad de ventanas de SQL:1999 es mucho más rica y puede utilizarse para escribir consultas bastante complejas con poco esfuerzo.

[4] Si el número total de tuplas de una partición no es divisible por *n*, el número de tuplas de cada cajón puede variar como mucho en 1. Las tuplas con el mismo valor del atributo de ordenación pueden asignarse cajones diferentes, de manera no determinista, para hacer igual el número de tuplas de cada cajón.

22.3. RECOPILACIÓN DE DATOS

El término **recopilación de datos** (**data mining**) hace referencia vagamente al proceso de análisis semiautomático de bases de datos de gran tamaño para hallar estructuras útiles. Al igual que la búsqueda de conocimiento en la inteligencia artificial (también denominada aprendizaje de la máquina), o el análisis estadístico, la recopilación de datos intenta descubrir reglas y estructuras a partir de los datos. No obstante, la recopilación de datos se diferencia del aprendizaje de la máquina y de la estadística en que trata con grandes volúmenes de datos, almacenados sobre todo en disco. Es decir, la recopilación de datos trata de la «búsqueda de conocimiento en las bases de datos».

Algunos tipos de conocimiento descubiertos a partir de una base de datos pueden representarse por un conjunto de **reglas**. A continuación se ofrece un ejemplo de regla, formulada de manera informal: «Las mujeres jóvenes con ingresos anuales superiores a 50.000 € son las personas que con mayor probabilidad compran coches deportivos de pequeño tamaño». Por supuesto, estas reglas no son verdaderas de modo universal, y tienen grados de «soporte» y de «confianza», como se verá. Otros tipos de conocimiento se representan por ecuaciones que relacionan entre sí diferentes variables, o mediante otros mecanismos de predicción de resultados cuando se conocen los valores de algunas variables.

Hay gran variedad de tipos posibles de estructuras que pueden resultar útiles, y se emplean diferentes técnicas para hallar tipos diferentes de estructuras. Se estudiarán unos cuantos ejemplos de estructuras y se verá el modo en que pueden obtenerse de manera automática de las bases de datos.

Suele haber una parte manual en la recopilación de datos, que consiste en el preprocesamiento de los datos hasta una forma aceptable para los algoritmos, y en el posprocesamiento de las estructuras descubiertas para hallar otras nuevas que puedan resultar útiles. También puede haber más de un tipo de estructura que se pueda descubrir a partir de una base de datos dada, y puede que se necesite la interacción manual para escoger los tipos de estructuras útiles. Por este motivo, la recopilación de datos es realmente un proceso semiautomático en la vida real. No obstante, la descripción que sigue se centrará en el aspecto automático de la recopilación.

22.3.1. Aplicaciones de la recopilación de datos

La información hallada tiene numerosas aplicaciones. Las aplicaciones más utilizadas son las que necesitan algún tipo de **predicción**. Por ejemplo, cuando una persona solicita una tarjeta de crédito, la compañía emisora quiere predecir si la persona constituye un buen riesgo de crédito. La predicción tiene que basarse en los atributos conocidos de la persona, como la edad, sus ingresos, sus deudas y su historial de pago de deudas. Las reglas para realizar la predicción se deducen de los mismos atributos de titulares de tarjetas de crédito pasados y actuales, junto con su conducta observada, como puede ser si han dejado de pagar los cargos de su tarjeta de crédito. Entre otros tipos de predicción están la predicción de los consumidores que pueden pasarse a un competidor (puede que se ofrezca a esos consumidores descuentos especiales para intentar que no se cambien), la predicción de la gente que puede responder a correo publicitario («correo basura») o la predicción de los tipos de empleo de tarjetas telefónicas que pueden resultar fraudulentos.

Otra clase de aplicaciones busca **asociaciones**, por ejemplo, los libros que se suelen comprar juntos. Si un cliente compra un libro, puede que la librería en línea le sugiera otros libros asociados. Si una persona compra una cámara, puede que el sistema sugiera accesorios que suelen comprarse junto a las cámaras. Un buen vendedor está atento a esas tendencias y las aprovecha para realizar más ventas. El desafío consiste en automatizar el proceso. Puede que otros tipos de asociación lleven al descubrimiento de relaciones causa-efecto. Por ejemplo, el descubrimiento de asociaciones inesperadas entre un medicamento recién introducido y los problemas cardiacos llevó al hallazgo de que el medicamento puede causar problemas cardiacos en algunas personas. El medicamento se retiró del mercado.

Las asociaciones son un ejemplo de **patrones descriptivos**. Las **agrupaciones** son otro ejemplo de este tipo de patrones. Por ejemplo, hace más de un siglo se descubrió una agrupación de casos de fiebre tifoidea alrededor de un pozo, lo que llevó al descubrimiento de que el agua del pozo estaba contaminada y estaba difundiendo la fiebre tifoidea. La detección de agrupaciones de enfermedades sigue siendo importante hoy en día.

22.3.2. Clasificación

Como ya se mencionó en el Apartado 22.3.1, la predicción es uno de los tipos más importantes de recopilación de datos. Se describirá lo que es la clasificación, se estudiarán técnicas para la creación de un tipo de clasificadores, denominados clasificadores de árboles de decisión, y se estudiarán otras técnicas de predicción.

De manera abstracta, el problema de la **clasificación** es el siguiente: dado que los elementos pertenecen a una de las clases, y dados los casos pasados (denominados **ejemplos de formación**) de los elementos junto con las clases a las que pertenecen, el problema es predecir la clase a la que pertenece un elemento nuevo. La clase del caso nuevo no se conoce, por lo que hay que utilizar los demás atributos del caso para predecir la clase.

La clasificación se puede llevar a cabo hallando reglas que dividan los datos dados en grupos disjuntos. Por

ejemplo, supóngase que una compañía de tarjetas de crédito quiera decidir si debe conceder una tarjeta a un solicitante. La compañía tiene amplia información sobre esa persona, como puede ser su edad, su nivel educativo, sus ingresos anuales y sus deudas actuales, la cual puede utilizar para adoptar una decisión.

Parte de esta información puede ser importante para el valor de crédito del solicitante, mientras que puede que otra parte no lo sea. Para adoptar la decisión la compañía asigna un nivel de valor de crédito de excelente, bueno, mediano o malo a cada integrante de un conjunto de muestra de clientes *actuales* según su historial de pagos. Luego, la compañía intenta hallar reglas que clasifiquen a sus clientes actuales como excelentes, buenos, medianos o malos con base en la información sobre esas personas diferente de su historial de pagos actual (que no está disponible para los clientes nuevos). Considérense sólo dos atributos: el nivel educativo (la titulación más alta conseguida) y los ingresos. Las reglas pueden ser de la forma siguiente:

∀ *persona P, P.titulación* = *máster* **and** *P.ingresos*
 > 75000 ⇒ *P.crédito* = *excelente*
∀ *persona P, P.titulación* = *bachiller* **or**
 (*P.ingresos* ≥ 25000 **and** *P.ingresos* ≤ 75000)
 ⇒ *P.crédito* = *bueno*

También aparecen reglas parecidas para los demás niveles de valor de crédito (mediano y malo).

El proceso de creación de clasificadores comienza con una muestra de los datos, denominada **conjunto de formación**. Para cada tupla del conjunto de formación ya se conoce la clase a la que pertenece. Por ejemplo, el conjunto de formación de las solicitudes de tarjetas de crédito pueden ser los clientes ya existentes, con su valor de crédito determinado a partir de su historial de pagos. Los datos actuales, o población, pueden consistir en toda la gente, incluida la que no es todavía cliente. Hay varias maneras de crear clasificadores, como se verá.

22.3.2.1. Clasificadores de árboles de decisión

Los clasificadores de árboles de decisión son una técnica muy utilizada para la clasificación. Como sugiere el nombre, los **clasificadores de árboles de decisión** utilizan un árbol; cada nodo hoja tiene una clase asociada, y cada nodo interno tiene un predicado (o, de manera más general, una función) asociado. La Figura 22.6 muestra un ejemplo de árbol de decisión.

Para clasificar un nuevo caso se empieza por la raíz y se recorre el árbol hasta alcanzar una hoja; en los nodos internos se evalúa el predicado (o la función) para el ejemplo de datos, para hallar a qué nodo hijo hay que ir. El proceso continúa hasta que se llega a un nodo hoja. Por ejemplo, si el nivel académico de la persona es de máster y sus ingresos son de 40K, partiendo de la raíz se sigue el arco etiquetado «máster», y desde allí el arco etiquetado «25K a 75K», hasta alcanzar una hoja. La clase de la hoja es «bueno», por lo que se puede predecir que el riesgo de crédito de esa persona es bueno.

Creación de clasificadores de árboles de decisión

La pregunta que se plantea es el modo de crear un clasificador de árboles de decisión, dado un conjunto de casos de formación. La manera más frecuente de hacerlo es utilizar un algoritmo **impaciente**, que trabaja de manera recursiva, comenzando por la raíz y constru-

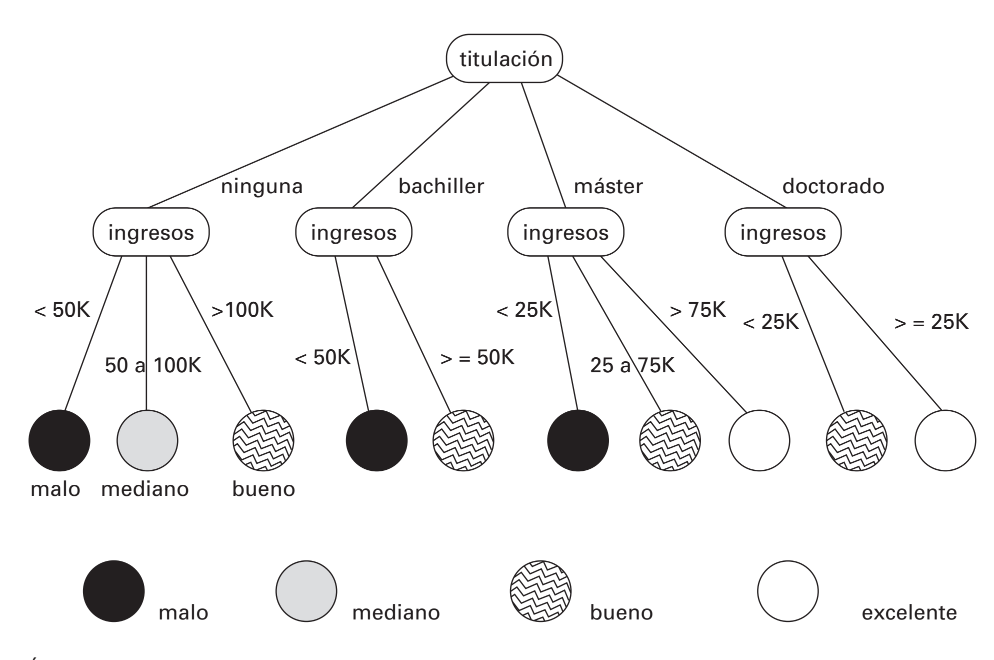


FIGURA 22.6. Árbol de clasificación.

yendo el árbol hacia abajo. Inicialmente sólo hay un nodo, la raíz, y todos los casos de formación están asociados con ese nodo.

En cada nodo, si todos o «casi todos» los ejemplos de formación asociados con el nodo pertenecen a la misma clase, el nodo se convierte en un nodo hoja asociado con esa clase. En caso contrario, hay que seleccionar un **atributo de partición** y **condiciones de partición** para crear nodos hijo. Los datos asociados con cada nodo hijo son el conjunto de ejemplos de formación que satisfacen la condición de partición de ese nodo hijo. En el ejemplo elegido, se escoge el atributo *titulación* y se crean cuatro hijos, uno por cada valor de la titulación. Las condiciones para los cuatro nodos hijo son *titulación* = ninguna, *titulación* = bachiller, *titulación* = máster y *titulación* = doctorado, respectivamente. Los datos asociados con cada hijo son los ejemplos de formación asociados con ese hijo. En el nodo correspondiente a máster se escoge el atributo *ingreso* con el rango de valores dividido en los intervalos 0 a 25.000, 25.000 a 50.000, 50.000 a 75.000 y más de 75.000. Los datos asociados con cada nodo son los ejemplos de formación con atributo *titulación* igual a máster y el atributo *ingresos* en cada uno de los rangos, respectivamente. Como optimización, ya que la clase para el rango de 25.000 a 50.000 y el rango de 50.000 a 75.000 es el mismo bajo el nodo *titulación* = máster, se han unido los dos rangos en uno solo que va de 25.000 a 75.000.

Las mejores particiones

De manera intuitiva, al escoger una secuencia de atributos de partición, se comienza con el conjunto de todos los ejemplos de formación, que es «impuro» en el sentido de que contiene ejemplos de muchas clases, y se acaba con las hojas, que son «puras» en el sentido de que en cada hoja todos los ejemplos de formación pertenecen a una única clase. Se verá brevemente el modo de medir cuantitativamente la pureza. Para evaluar la ventaja de escoger un atributo concreto y la condición para la partición de los datos en un nodo se mide la pureza de los datos en los hijos resultantes de la partición según ese atributo. Se escogen el atributo y la condición que producen la pureza máxima.

La pureza de un conjunto S de ejemplos de formación puede medirse cuantitativamente de varias maneras. Supóngase que hay k clases y que de los ejemplos en S la fracción de ejemplos de la clase i es p_i. Una medida de pureza, la **medida de Gini,** se define como

$$\text{Gini}(S) = 1 - \sum_{i-1}^{k} p_i^2$$

Cuando todos los ejemplos están en una sola clase, el valor de Gini es 0, mientras que alcanza su máximo (de $1 - 1/k$) si cada clase tiene el mismo número de ejemplos. Otra medida de la pureza es la **medida de la entropía**, que se define como

$$\text{Entropía}(S) = -\sum_{i-1}^{k} p_i \log_2 p_i$$

El valor de la entropía es 0 si todos los ejemplos están en una sola clase y alcanza su máximo cuando cada clase tiene el mismo número de ejemplos. La medida de la entropía proviene de la teoría de la información.

Cuando un conjunto S se divide en varios conjuntos S_i, $i = 1, 2, \ldots, r$, se puede medir la pureza del conjunto de conjuntos resultante como:

$$\text{Pureza}(S_1, S_2, \ldots, S_r) = \sum_{i=1}^{r} \frac{|S_i|}{|S|} \text{pureza}(S_i)$$

Es decir, la pureza es la media ponderada de la pureza de los conjuntos S_i. La fórmula anterior puede utilizarse tanto con la medida de la pureza de Gini como con la medida de la pureza de la entropía.

La **ganancia de información** debida a una partición concreta de S en S_i, $i = 1, 2, \ldots, r$ es, entonces,

$$\begin{aligned}&\text{Ganancia-información}(S, \{S_1, S_2, \ldots, S_r\}) \\ &= \text{pureza}(S) - \text{pureza}(S_1, S_2, \ldots, S_r)\end{aligned}$$

Las particiones en menor número de conjuntos son preferibles a las particiones en muchos conjuntos, ya que llevan a árboles de decisión más sencillos y significativos. El número de elementos en cada uno de los conjuntos S_i también puede tenerse en cuenta; en caso contrario, que un conjunto S_i tenga 0 elementos o 1 elemento supondría una gran diferencia en el número de conjuntos, aunque la partición fuera la misma para casi todos los elementos. El **contenido de información** de una partición concreta puede expresarse en términos de entropía como

$$\begin{aligned}&\text{Contenido-información}(S, \{S_1, S_2, \ldots, S_r\})) = \\ &= -\sum_{i-1}^{r} \frac{|Si|}{|S|} \log_2 \frac{|Si|}{|S|}\end{aligned}$$

Todo esto lleva a una definición: La **mejor partición** para un atributo es la que da el máximo **índice de ganancia de información**, definido como

$$\frac{\text{Ganancia-información}(S, \{S_1, S_2, \ldots, S_r\})}{\text{Contenido-información}(S, \{S_1, S_2, \ldots, S_r\})}$$

Búsqueda de las mejores particiones

Hay que averiguar el modo de hallar la mejor partición para un atributo. El modo de dividir un atributo depende del tipo de atributo. Los atributos pueden tener **valores continuos**, es decir, los valores se pueden ordenar de manera significativa para la clasificación, como la edad o los ingresos, o pueden ser **categóricos**, es decir, no tener ningún orden significativo, como los nombres de los departamentos o los de los países. No se espera que el orden de los nombres de los departamentos o el

de los países tenga ningún significado para la clasificación.

Generalmente, los atributos que son números (enteros o reales) se tratan como valores continuos y los atributos de cadenas de caracteres se tratan como categóricos, pero esto puede controlarlo el usuario del sistema. En el ejemplo escogido se ha tratado el atributo *titulación* como categórico y el atributo *ingresos* como valor continuo.

En primer lugar se considera el modo de hallar las mejores particiones para los atributos con valores continuos. Por sencillez sólo se considerarán **particiones binarias** de los atributos con valores continuos, es decir, particiones que den lugar a dos hijos. El caso de las **particiones múltiples** es más complicado; véanse las notas bibliográficas para hallar referencias sobre este asunto.

Para hallar la mejor partición binaria de un atributo con valores continuos, en primer lugar, se ordenan los valores del atributo en los ejemplos de formación. Luego se calcula la ganancia de información obtenida por la división en cada valor. Por ejemplo, si los ejemplos de formación tienen los valores 1, 10, 15 y 25 para un atributo, los puntos de partición considerados son 1, 10 y 15; en cada caso, los valores menores o iguales que el punto de partición forman una partición y el resto de los valores forman la otra. La mejor partición binaria para el atributo es la partición que da la ganancia de información máxima.

Para los atributos categóricos se pueden tener particiones múltiples, con un hijo para cada valor del atributo. Esto funciona muy bien para los atributos categóricos con pocos valores diferentes, como la titulación o el sexo. No obstante, si el atributo tiene muchos valores diferentes, como los nombres de los departamentos en compañías grandes, la creación de un hijo para cada valor no es una buena idea. En esos casos se procura combinar varios valores en cada hijo para crear un número menor de hijos. Véanse las notas bibliográficas para hallar referencias al modo de hacerlo.

Algoritmo de construcción del árbol de decisión

La idea principal de la construcción de árboles de decisión es la evaluación de los diferentes atributos y de las distintas condiciones de partición y la selección del atributo y de la condición de partición que generen el índice máximo de ganancia de información. El mismo procedimiento funciona de manera recursiva en cada uno de los conjuntos resultantes de la partición, lo que hace que se construya de manera recursiva el árbol de decisión. Si los datos pueden clasificarse de manera perfecta, la recursión se detiene cuando la pureza de un conjunto sea 0. No obstante, los datos suelen tener ruido, o puede que un conjunto sea tan pequeño que no se justifique estadísticamente su partición. En ese caso, la partición se detiene cuando la pureza del conjunto es «bastante alta» y la clase de la hoja resultante se define como la clase de la mayoría de los elementos del conjunto. En general, las diferentes ramas del árbol pueden crecer hasta niveles diferentes.

La Figura 22.7 muestra pseudocódigo para un procedimiento recursivo de construcción de un árbol, que toma al conjunto de ejemplos de formación *S* como parámetro. La recursión se detiene cuando el conjunto es lo bastante puro o el conjunto *S* es demasiado pequeño para que más particiones resulten estadísticamente significativas. Los parámetros δ_p y δ_t definen los valores de corte para la pureza y el tamaño; puede que el sistema les dé valores predeterminados, que los usuarios pueden cancelar.

Hay gran variedad de algoritmos de construcción de árboles de decisión y se esbozarán las características distintivas de unos cuantos. Véanse las notas bibliográficas para hallar más detalles. Con conjuntos de datos de tamaño muy grande la realización de particiones puede resultar muy costosa, ya que implica la realización repetida de copias. Por tanto, se han desarrollado varios algoritmos para minimizar el coste de E/S y el coste de computación cuando los datos de formación son mayores que la memoria disponible.

Varios de los algoritmos también podan los subárboles del árbol de decisión generado para reducir el **exceso de ajuste**: un subárbol tiene exceso de ajuste si se ha ajustado tanto a los detalles de los datos de formación que comete muchos errores de clasificación con otros datos. Se poda un subárbol sustituyéndolo por un nodo hoja. Hay varias heurísticas de poda; una utiliza parte de los datos de formación para construir el árbol y otra parte para comprobarlo. La heurística poda el subárbol si descubre que los errores de clasificación de los casos de prueba se reducirían si se sustituyera por un nodo hoja.

Se pueden generar reglas de clasificación a partir de los árboles de decisión, si se desea. Para cada hoja se genera una regla de la manera siguiente: la parte izquierda es la conjunción de todas las condiciones de partición del camino hasta la hoja y la clase es la clase de la mayoría de los ejemplos de formación de la hoja. Un ejemplo de estas reglas de clasificación es

titulación = *máster* **and** *ingresos* > 75.000 ⇒ *excelente*

```
procedure CultivarÁrbol(S)
    Partición(S);

procedure Partición (S)
    if (pureza(S) > δp or |S| < δt) then
        return;
    for each atributo A
        evalúa las particiones según el atributo A;
    Utiliza la mejor partición hallada (para todos los atributos)
                para dividir S en S1, S2,..., Sr;
    for i = 1, 2,..., r
        Partición(Si);
```

FIGURA 22.7. Construcción recursiva de un árbol de decisión.

22.3.2.2. Otros tipos de clasificadores

Hay varios tipos de clasificadores aparte de los clasificadores de árbol. Dos tipos que han resultado bastante útiles son los *clasificadores de redes neuronales* y los *clasificadores bayesianos*. Los clasificadores de redes neuronales utilizan los datos de formación para adiestrar redes neuronales artificiales. Hay gran cantidad de literatura sobre las redes neuronales y aquí no se hablará más de ellas.

Los **clasificadores bayesianos** hallan la distribución de los valores de los atributos para cada clase de los datos de formación; cuando se da un nuevo caso, d, utilizan la información de la distribución para estimar, para cada clase c_j, la probabilidad de que el caso d pertenezca a la clase c_j, denotada por $p(c_j/d)$, de la manera que aquí se describe. La clase con la probabilidad máxima se transforma en la clase predicha para el caso d.

Para hallar la probabilidad $p(c_j/d)$ de que el caso d esté en la clase c_j los clasificadores bayesianos utilizan el **teorema de Bayes**, que dice

$$p(c_j/d) = \frac{p(d/c_j)p(c_j)}{p(d)}$$

donde $p(d/c_j)$ es la probabilidad de que se genere el caso d dada la clase c_j, $p(c_j)$ es la probabilidad de ocurrencia de la clase c_j y $p(d)$ es la probabilidad de que ocurra el caso d. De éstas, $p(d)$ puede ignorarse, ya que es igual para todas las clases. $p(c_j)$ no es más que la fracción de los casos de formación que pertenecen a la clase c_j.

Hallar exactamente $p(d/c_j)$ resulta difícil, ya que exige una distribución completa de los casos de c_j. Para simplificar la tarea los **clasificadores bayesianos ingenuos** dan por hecho que los atributos tienen distribuciones independientes y, por tanto, estiman

$$p(d/c_j) = p(d_1|c_j) * p(d_2|c_j) * \ldots * p(d_n/c_j)$$

Es decir, la probabilidad de que ocurra el caso d es el producto de la probabilidad de ocurrencia de cada uno de los valores *di* del atributo d, dado que la clase es c_j.

Las probabilidades $p(d_i/c_j)$ proceden de la distribución de los valores de cada atributo i, para cada clase c_j. Esta distribución se calcula a partir de los ejemplos de formación que pertenecen a cada clase c_j; la distribución suele aproximarse mediante un histograma. Por ejemplo, se puede dividir el rango de valores del atributo i en intervalos iguales y almacenar la fracción de casos de la clase c_j que caen en cada intervalo. Dado un valor d_i para el atributo i, el valor de $p(d_i/c_j)$ es simplemente la fracción de casos que pertenecen a la clase c_j que caen en el intervalo al que pertenece d_i.

Una ventaja significativa de los clasificadores bayesianos es que pueden clasificar los casos con valores de los atributos desconocidos y nulos; los atributos desconocidos o nulos simplemente se omiten del cálculo de probabilidades. Por el contrario, los clasificadores de árboles de decisión no pueden tratar de manera significativa las situaciones en que el caso que hay que clasificar tiene un valor nulo para el atributo de partición utilizado para avanzar por el árbol de decisión.

22.3.2.3. Regresión

La **regresión** trata de la predicción de valores, no de clases. Dados los valores de un conjunto de variables, $X_1, X_2, \ldots, X_n$, se desea predecir el valor de una variable Y. Por ejemplo, se puede tratar el nivel educativo como un número y los ingresos como otro número y, con base en estas dos variables, querer predecir la posibilidad de impago, que podría ser un porcentaje de probabilidad de impago o el importe impagado.

Una manera de inferir los coeficientes $a_0, a_1, a_2, \ldots, a_n$ tales que

$$Y = a_0 + a_1 * X_1 + a_2 * X_2 + \ldots + a_n * X_n$$

La búsqueda de ese polinomio lineal se denomina **regresión lineal**. En general, se quiere hallar una curva (definida por un polinomio o por otra fórmula) que se ajuste a los datos; el proceso también se denomina **ajuste de la curva**.

El ajuste sólo puede ser aproximado, debido al ruido de los datos o a que la relación no sea exactamente un polinomio, por lo que la regresión pretende hallar coeficientes que den el mejor ajuste posible. Hay técnicas estándar en estadística para hallar los coeficientes de regresión. Aquí no se estudiarán esas técnicas, pero las notas bibliográficas ofrecen referencias.

22.3.3. Reglas de asociación

Los comercios minoristas suelen estar interesados en las **asociaciones** entre los diferentes artículos que compra la gente. Ejemplos de esas asociaciones son:

- Alguien que compra pan es bastante probable que compre también leche.
- Una persona que compró el libro *Fundamentos de bases de datos* es bastante probable que también compre el libro *Fundamentos de sistemas operativos*.

La información de asociación puede utilizarse de varias maneras. Cuando un cliente compra un libro determinado puede que la librería en línea le sugiera los libros asociados. Puede que la tienda de alimentación decida colocar el pan cerca de la leche, ya que suelen comprarse juntos, para ayudar a los clientes a hacer la compra más rápidamente. O puede que la tienda los coloque en extremos opuestos del mostrador y coloque otros artículos asociados entre medias para inducir a la gente a comprar también esos artículos, mientras los clientes van de un extremo a otro del mostrador. Puede que una tienda que ofrece descuento en un artículo asociado no lo ofrezca en el otro, ya que, de todos modos, el cliente comprará el segundo artículo.

Reglas de asociación

Un ejemplo de regla de asociación es

$$pan \Rightarrow leche$$

En el contexto de las compras de alimentación, la regla dice que los clientes que compran pan también tienden a comprar leche con una probabilidad elevada. Una regla de asociación debe tener una **población** asociada: la población consiste en un conjunto de **casos**. En el ejemplo de la tienda de alimentación, la población puede consistir en todas las compras en la tienda de alimentación; cada compra es un caso. En el caso de una librería, la población puede consistir en toda la gente que realiza compras, independientemente del momento en que las hayan realizado. Cada consumidor es un caso. Aquí, el analista ha decidido que el momento de realización de la compra no es significativo, mientras que, para el ejemplo de la tienda de alimentación, puede que el analista haya decidido concentrarse en cada compra, ignorando las diferentes visitas de un mismo cliente.

Las reglas tienen un *soporte*, así como una *confianza* asociados. Los dos se definen en el contexto de la población:

- El **soporte** es una medida de la fracción de la población que satisface tanto el antecedente como el consecuente de la regla.

  Por ejemplo, supóngase que sólo el 0.001 por ciento de todas las compras incluyen leche y destornilladores. El soporte de la regla

  $$leche \Rightarrow destornilladores$$

  es bajo. Puede que la regla ni siquiera sea estadísticamente significativa —quizás solo hubiera una única compra que incluyera leche y destornilladores. Las empresas no suelen estar interesadas en las reglas que tienen un soporte bajo, ya que afectan a pocos clientes y no merece la pena prestarles atención.

  Por otro lado, si el 50 por ciento de las compras implica leche y pan, el soporte de las reglas que afecten al pan y a la leche (y a ningún otro artículo) es relativamente elevado, y puede que merezca la pena prestarles atención. El grado mínimo de soporte que se considera deseable exactamente depende de la aplicación.

- La **confianza** es una medida de la frecuencia con que el consecuente es cierto cuando lo es el antecedente. Por ejemplo, la regla

  $$pan \Rightarrow leche$$

  tiene una confianza del 80 por ciento si el 80 por ciento de las compras que incluyen pan incluyen también leche. Las reglas con una confianza baja no son significativas. En las aplicaciones comerciales las reglas suelen tener confianzas significativamente menores del 100 por ciento, mientras que en otros campos, como la física, las reglas pueden tener confianzas elevadas.

  Hay que tener en cuenta que la confianza de *pan* ⇒ *leche* puede ser muy diferente de la confianza de *leche* ⇒ *pan*, aunque las dos tienen el mismo soporte.

Búsqueda de reglas de asociación

Para descubrir reglas de asociación de la forma

$$i_1, i_2, \ldots, i_n \Rightarrow i_0$$

primero hay que hallar conjuntos de elementos con soporte suficiente, denominados **conjuntos grandes de elementos**. En el ejemplo que se trata se hallan conjuntos de elementos que están incluidos en un número de casos lo bastante grande. En breve se verá el modo de calcular conjuntos grandes de elementos.

Para cada conjunto grande de elementos se obtienen todas las reglas con confianza suficiente que afecten a todos los elementos del conjunto y sólo a ellos. Para cada conjunto grande de elementos S se obtiene una regla $S - s \Rightarrow s$ para cada subconjunto $s \subset S$, siempre que $S - s \Rightarrow s$ tenga confianza suficiente; la confianza de la regla la da el soporte de s dividido por el soporte de S.

Ahora se considerará el modo de generar todos los conjuntos grandes de elementos. Si el número de conjuntos de elementos posibles es pequeño, basta con un solo paso por los datos para detectar el nivel de soporte de todos los conjuntos. Se lleva una cuenta, con valor inicial 0, para cada conjunto de elementos. Cuando se captura el registro de una compra, la cuenta se incrementa para cada conjunto de elementos tal que todos los elementos del conjunto estén contenidos en la compra. Por ejemplo, si una compra incluye los elementos a, b y c, se incrementará el contador para $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{b, c\}$, $\{a, c\}$ y $\{a, b, c\}$. Los conjuntos con un contador lo bastante elevado al final del pase se corresponden con los elementos que tienen un grado de asociación elevado.

El número de conjuntos crece de manera exponencial, lo que hace inviable el proceso que se acaba de describir si el número de elementos es elevado. Afortunadamente, casi todos los conjuntos tienen normalmente un soporte muy bajo; se han desarrollado optimizaciones para no considerar la mayor parte de esos conjuntos. Estas técnicas utilizan varios pases por la base de datos y sólo consideran algunos conjuntos en cada pase.

En la técnica **a priori** para la generación de conjuntos de artículos grandes sólo se consideran en el primer pase los conjuntos con un solo elemento. En el segundo pase se consideran los conjuntos con dos artículos, etcétera.

Al final de cada pase todos los conjuntos con soporte suficiente se consideran conjuntos grandes de elementos. Los conjuntos que se ha hallado que tienen demasiado poco soporte al final de cada pase se elimi-

nan. Una vez eliminado un conjunto no hace falta considerar ninguno de sus superconjuntos. En otros términos, en el pase *i* sólo hay que contar el soporte de los conjuntos de tamaño *i* tales que se haya hallado que todos sus subconjuntos tienen un soporte lo bastante elevado; basta con probar todos los subconjuntos de tamaño $i - 1$ para asegurase de que se cumple esta propiedad. Al final del pase *i* se halla que ningún conjunto de tamaño *i* tiene el soporte suficiente, por lo que no hace falta considerar ningún conjunto de tamaño $i + 1$. Entonces, el cálculo se termina.

22.3.4. Otros tipos de asociación

El uso de meras reglas de asociación tiene varios inconvenientes. Uno de los principales es que muchas asociaciones no son muy interesantes, ya que pueden predecirse. Por ejemplo, si mucha gente compra cereales y mucha gente compra pan, se puede predecir que un número bastante grande de personas comprará las dos cosas, aunque no haya ninguna relación entre las dos compras. Lo que resultaría interesante es una **desviación** de la ocurrencia conjunta de las dos compras. En términos estadísticos, se buscan **correlaciones** entre los artículos; las correlaciones pueden ser positivas, en las que la ocurrencia conjunta es superior a lo esperado, o negativa, en la que los elementos ocurren conjuntamente menos frecuentemente de lo predicho. Se puede consultar cualquier libro de texto estándar de estadística para hallar más información sobre las correlaciones.

Otra clase importante de aplicaciones de recopilación de datos son las asociaciones de secuencias (o correlaciones). Las series de datos temporales, como las cotizaciones bursátiles en una serie de días, constituyen un ejemplo de datos de secuencias. Los analistas bursátiles desean hallar asociaciones entre las secuencias de cotizaciones. Un ejemplo de asociación de este tipo es la regla siguiente: «Siempre que las tasas de interés de los bonos suben, las cotizaciones bursátiles bajan en un plazo de dos días». El descubrimiento de esta asociación entre secuencias puede ayudar a adoptar decisiones de inversión inteligentes. Véanse las notas bibliográficas para hallar referencias a la investigación en este campo.

Las desviaciones de las estructuras temporales suelen resultar interesantes. Por ejemplo, si una empresa ha estado creciendo a una tasa constante cada año, una desviación de la tasa de crecimiento habitual resulta sorprendente. Si las ventas de ropa de invierno bajan en verano, ya que puede predecirse con base en los años anteriores, una desviación que no se pudiera predecir a partir de la experiencia pasada se consideraría interesante. Las técnicas de recopilación pueden hallar desviaciones de lo esperado con base en las estructuras temporales o secuenciales pasadas. Véanse las notas bibliográficas para hallar referencias a la investigación en este campo.

22.3.5. Agrupamiento

De manera intuitiva, el agrupamiento hace referencia al problema de hallar agrupaciones de puntos en los datos dados. El problema del **agrupamiento** puede formalizarse de varias maneras a partir de las métricas de distancias. Una manera es formularlo como el problema de agrupar los puntos en *k* conjuntos (para un *k* dado) de modo que la distancia media de los puntos al *centroide* de su agrupación asignada sea mínima[5]. Otra manera es agrupar los puntos de modo que la distancia media entre cada par de puntos de cada agrupación sea mínima. Hay otras definiciones; véanse las notas bibliográficas para hallar más detalles. Pero la intuición subyacente a todas estas definiciones es agrupar los puntos parecidos en un único conjunto.

Otro tipo de agrupamiento aparece en los sistemas de clasificaciones de la biología. (Esos sistemas de clasificación no intentan *predecir* las clases, sino agrupar los elementos relacionados.) Por ejemplo, los leopardos y los seres humanos se agrupan bajo la clase mamíferos, mientras que los cocodrilos y las serpientes se agrupan bajo los reptiles. Tanto los mamíferos como los reptiles están bajo la clase común de los cordados. La agrupación de los mamíferos tiene subagrupaciones, como los carnívoros y los primates. Por tanto, se tiene un **agrupamiento jerárquico**. Dadas las características de las diferentes especies, los biólogos han creado un esquema complejo de agrupamiento jerárquico que agrupa las especies relacionadas en diferentes niveles de la jerarquía.

El agrupamiento jerárquico también resulta útil en otros dominios; para agrupar documentos, por ejemplo. Los sistemas de directorio de Internet (como el de Yahoo) agrupan los documentos relacionados de manera jerárquica (véase el Apartado 22.5.5). Los algoritmos de agrupamiento jerárquico pueden clasificarse como algoritmos de **agrupamiento aglomerativo**, que comienzan creando agrupaciones pequeñas y luego crean los niveles superiores, o como algoritmos de **agrupamiento divisivo**, que primero crean los niveles superiores del agrupamiento jerárquico y luego refinan cada agrupación resultante en agrupaciones de niveles inferiores.

[5] El centroide de un conjunto de puntos se define como un punto cuyas coordenadas en cada dimensión son el promedio de las coordenadas de todos los puntos de ese conjunto en esa dimensión. Por ejemplo, en dos dimensiones, el centroide de un conjunto de puntos $\{ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \}$ viene dado por $\left(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n}\right)$.

La comunidad estadística ha estudiado extensamente los agrupamientos. La investigación en bases de datos ha proporcionado algoritmos escalables de agrupamiento que pueden agrupar conjuntos de datos de tamaño muy grande (que puede que no quepan en la memoria). El algoritmo de agrupamiento Birch es un algoritmo de este tipo. De manera intuitiva, los puntos de datos se insertan en una estructura arbórea multidimensional (basada en los árboles R descritos en el Apartado 23.3.5.3) y son llevados a los nodos hoja correspondientes de acuerdo con su cercanía a los puntos representativos de los nodos internos del árbol. Los puntos próximos, por tanto, se agrupan en los nodos hoja, y se resumen si hay más puntos de los que caben en la memoria. El procesamiento posterior a la inserción de todos los puntos da el agrupamiento global deseado. Véanse las notas bibliográficas para hallar las referencias al algoritmo Birch y a otras técnicas de agrupamiento, incluidos los algoritmos para el agrupamiento jerárquico.

Una aplicación interesante del agrupamiento es la predicción de las películas nuevas (o de los libros nuevos, o de la música nueva) que es probable que interesen a una persona, con base en:

1. Las preferencias cinematográficas pasadas de esa persona
2. Otras personas con preferencias pasadas parecidas
3. Las preferencias de esa gente entre las películas nuevas

Un enfoque de este problema es el siguiente. Para hallar gente con preferencias anteriores parecidas se crean agrupaciones de personas de acuerdo con sus preferencias cinematográficas. La exactitud del agrupamiento puede mejorarse agrupando previamente las películas por su parecido, de modo que, aunque la gente no haya visto las mismas películas, se agruparán si han visto películas parecidas. Se puede repetir el agrupamiento, agrupando alternativamente gente y películas hasta que se alcance un equilibrio. Dado un nuevo usuario, se halla una agrupación de usuarios lo más parecidos posibles a él, con base en las preferencias del usuario por las películas que ya ha visto. Luego se predice que las películas de las agrupaciones de películas que son populares en la agrupación de ese usuario es probable que resulten interesantes para el nuevo usuario. De hecho, este problema es un caso de *filtrado colaborativo*, en el que los usuarios colaboran en la tarea de filtrado de la información para hallar información de interés.

22.3.6. Otros tipos de recopilación

Las técnicas de recopilación de datos para la **recopilación de texto** en documentos de texto. Por ejemplo, hay herramientas que forman agrupaciones de las páginas que ha visitado un usuario; esto ayuda a los usuarios cuando examinan su historial de exploración para hallar las páginas que han visitado anteriormente. La distancia entre las páginas puede basarse, por ejemplo, en las palabras frecuentes en esas páginas (véase el Apartado 22.5.1.3). Otra aplicación es la clasificación automática de las páginas en directorios Web, de acuerdo con su parecido con otras páginas (véase el Apartado 22.5.5).

Los sistemas de **visualización de datos** ayudan a los usuarios a examinar grandes volúmenes de datos y a detectar visualmente las estructuras. Las exhibiciones visuales de datos —como los mapas, los gráficos y otras representaciones gráficas— permiten que los datos se presenten a los usuarios de manera compacta. Una sola pantalla gráfica puede codificar tanta información como un número mucho mayor de pantallas de texto. Por ejemplo, si el usuario desea averiguar si los problemas de producción en las factorías están correlacionadas con su ubicación se pueden codificar las ubicaciones problemáticas en un color especial —por ejemplo, rojo— en un mapa. El usuario puede descubrir rápidamente las ubicaciones en las que se dan los problemas. El usuario puede así formular hipótesis sobre el motivo de que los problemas se produzcan en esas ubicaciones y verificarlas cuantitativamente con la base de datos.

Otro ejemplo más: la información sobre los valores puede codificarse como colores y mostrarse con sólo un píxel de área de pantalla. Para detectar las asociaciones entre pares de elementos se puede utilizar una matriz bidimensional de píxeles en la que cada fila y cada columna representen un elemento. El porcentaje de transacciones que compran los dos elementos puede codificarse por la intensidad del color de los píxeles. Los elementos con asociaciones elevadas aparecerán en la pantalla como píxeles brillantes —fáciles de detectar contra el fondo más oscuro.

Los sistemas de visualización de datos no detectan de manera automática las estructuras, sino que proporcionan soporte del sistema para que los usuarios las detecten. Dado que los seres humanos son muy buenos en la detección de estructuras visuales, la visualización de los datos es un componente importante de la recopilación de datos.

22.4. ALMACENAMIENTO DE DATOS

Las grandes empresas tienen presencia en muchos lugares, cada uno de los cuales puede generar un gran volumen de datos. Por ejemplo, las cadenas de tiendas minoristas tienen centenares o millares de tiendas, mientras que las compañías de seguros pueden tener datos de millares de oficinas locales. Además, las organizaciones grandes tienen una estructura compleja de organización interna y, por tanto, puede que los diferentes datos se hallen en ubicaciones diferentes, en sistemas operativos distintos o bajo esquemas diferentes. Por ejemplo, puede que los datos de los problemas de fabricación y los datos sobre las quejas de los clientes estén almacenados en diferentes sistemas de bases de datos. Los encargados de adoptar las decisiones empresariales necesitan tener acceso a la información de todas esas fuentes. La formulación de consultas a cada una de las fuentes es a la vez engorroso e ineficiente. Además, puede que los orígenes de datos sólo almacenen los datos actuales, mientras que es posible que los encargados de adoptar las decisiones empresariales necesiten tener acceso también a datos anteriores, por ejemplo, información sobre la manera en que se han modificado las pautas de compra el año pasado puede resultar de gran importancia. Los almacenes de datos proporcionan una solución a estos problemas.

Los **almacenes de datos** (**data warehouses**) son depósitos (o archivos) de información reunida de varios orígenes, almacenada bajo un esquema unificado en un solo sitio. Una vez reunida, los datos se almacenan mucho tiempo, lo que permite el acceso a datos históricos. Así, los almacenes de datos proporcionan a los usuarios una sola interfaz consolidada con los datos, lo que hace más fáciles de escribir las consultas de ayuda a la toma de decisiones. Además, al tener acceso a la información para la ayuda de la toma de decisiones desde un almacén de datos, el encargado de adoptar las decisiones se asegura de que los sistemas de procesamiento en línea de las transacciones no se vean afectados por la carga de trabajo de la ayuda de la toma de decisiones.

22.4.1. Componentes de los almacenes de datos

La Figura 22.8 muestra la arquitectura de un almacén de datos típico e ilustra la recogida de los datos, su almacenamiento y el soporte de las consultas y del análisis de datos. Entre los problemas que hay que resolver al crear un almacén de datos están los siguientes:

- **Momento y modo de la recogida de datos.** En una **arquitectura dirigida por los orígenes** para la recogida de los datos, los orígenes de los datos transmiten la información nueva, bien, de manera continua (a medida que se produce el procesamiento de las transacciones) o de manera periódica (de noche, por ejemplo). En una **arquitectura dirigida por el destino**, el almacén de datos envía de manera periódica solicitudes de datos nuevos a los orígenes de datos.

  A menos que las actualizaciones de los orígenes de datos se repliquen en el almacén de datos mediante un compromiso de dos fases, el almacén de datos nunca estará actualizado respecto a los orígenes de datos. El compromiso de dos fases suele resultar demasiado costoso para ser una opción aceptable, por lo que los almacenes de datos suelen tener datos ligeramente desactualizados. Eso, no obstante, no suele suponer un problema para los sistemas de ayuda a la toma de decisiones.

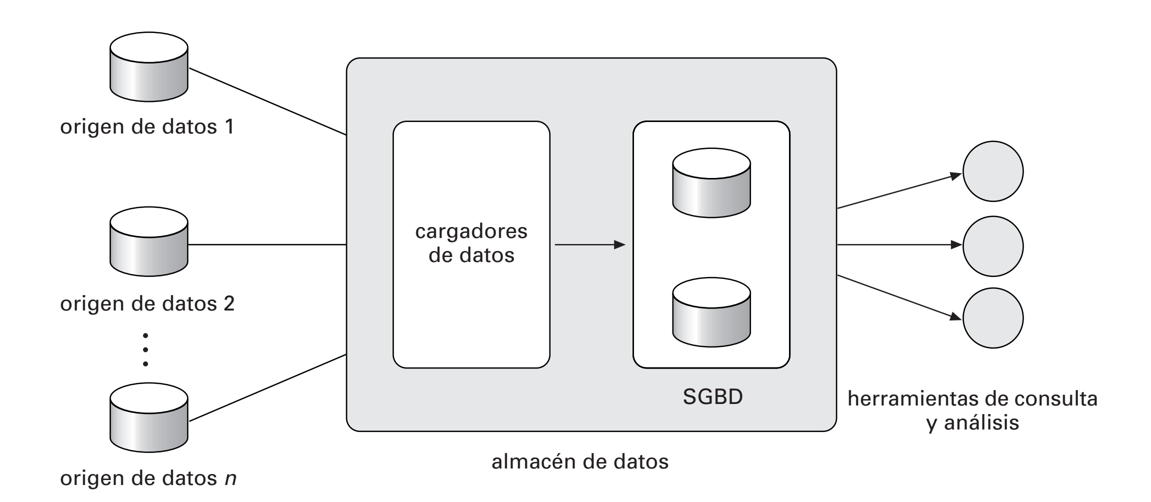


FIGURA 22.8. Arquitectura de los almacenes de datos.

- **Selección del esquema.** Es probable que los orígenes de datos que se han creado de manera independiente tengan esquemas diferentes. De hecho, puede que utilicen diferentes modelos de datos. Parte de la labor de los almacenes de datos es llevar a cabo la integración de los esquemas y convertir los datos al esquema integrado antes de almacenarlos. En consecuencia, los datos almacenados en el almacén de datos no son una mera copia de los datos de los orígenes de datos. Por el contrario, se pueden considerar como una vista materializada de los datos de los orígenes de datos.
- **Limpieza de los datos.** La labor de corregir y realizar un procesamiento previo de los datos se denomina **limpieza de los datos**. Los orígenes de datos suelen entregar datos con numerosas inconsistencias de carácter menor, que pueden corregirse. Por ejemplo, los nombres suelen estar mal escritos y puede que las direcciones tengan mal escritos los nombres de la calle, del distrito o de la ciudad, o puede que los códigos postales se hayan introducido de manera incorrecta. Esto puede corregirse en un grado razonable consultando una base de datos de los nombres de las calles y de los códigos postales de cada ciudad. Las listas de direcciones recogidas de varios orígenes pueden tener duplicados que haya que eliminar en una **operación de mezcla-purga**. Los registros de varias personas de una misma casa pueden agruparse para que sólo se realice a cada casa un envío de correo; esta operación se denomina **domiciliación**.
- **Propagación de las actualizaciones.** Las actualizaciones de las relaciones en los orígenes de datos deben propagarse a los almacenes de datos. Si las relaciones en los almacenes de datos son exactamente las mismas que en los orígenes de datos, la propagación es directa. En caso contrario, el problema de la propagación de las actualizaciones es básicamente el problema del *mantenimiento de las vistas* que se estudió en el Apartado 14.5.
- **Resúmenes de los datos.** Los datos brutos generados por un sistema de procesamiento de transacciones pueden ser demasiado grandes para almacenarlos en línea. No obstante, se pueden responder muchas consultas manteniendo únicamente datos resumen obtenidos por agregación de las relaciones, en lugar de mantener las relaciones enteras. Por ejemplo, en lugar de almacenar los datos de cada venta de ropa, se pueden almacenar las ventas totales de ropa por nombre de artículo y por categoría.

  Supóngase que una relación *r* ha sido sustituida por una relación resumen *s*. Todavía se puede permitir a los usuarios que planteen consultas como si la relación *r* estuviera disponible en línea. Si la consulta sólo necesita datos resumidos, puede que sea posible transformarla en una equivalente utilizando *s* en lugar de *r*; véase el Apartado 14.5.

22.4.2. Esquemas de los almacenes de datos

Los almacenes de datos suelen tener esquemas diseñados para el análisis de los datos y emplean herramientas como las herramientas OLAP. Por tanto, los datos suelen ser datos multidimensionales, con atributos de dimensión y atributos de medida. Las tablas que contienen datos multidimensionales se denominan **tablas de hechos** y suelen ser muy grandes. Las tablas que registran información de ventas de una tienda minorista, con una tupla para cada artículo a la venta, son un ejemplo típico de tablas de hechos. Las dimensiones de la tabla *ventas* incluyen lo que es el artículo (generalmente un identificador del artículo como el utilizado en los códigos de barras), la fecha en que se ha vendido, la ubicación (tienda) en que se vendió, el cliente que lo ha comprado, etcétera. Entre los atributos de medida pueden estar el número de artículos vendidos y el precio de cada artículo.

Para minimizar los requisitos de almacenamiento los atributos de dimensiones suelen ser identificadores breves que actúan de claves externas en otras tablas denominadas **tablas de dimensiones**. Por ejemplo, la tabla de hechos *ventas* tiene los atributos *id-artículo*, *id-tienda*, *id-cliente* y *fecha* y los atributos de medida *número* y *precio*. El atributo *id-tienda* es una clave externa en la tabla de dimensiones *tienda*, que tiene otros atributos como la ubicación de la tienda (ciudad, estado, país). El atributo *id-artículo* de la tabla *ventas* es una clave externa de la tabla de dimensiones *info-artículo*, que contiene información como el nombre del artículo, la categoría a la que pertenece el artículo y otros detalles del artículo como el color y la talla. El atributo *id-cliente* es una clave externa de la tabla *cliente*, que contiene atributos como el nombre y la dirección de los clientes. También se puede ver el atributo *fecha* como clave externa de la tabla *info-fecha*, que da el mes, el trimestre y el año de cada fecha.

El esquema resultante aparece en la Figura 22.9. Un esquema así, con una tabla de hechos, varias tablas de dimensiones y claves externas procedentes de la tabla de hechos en las tablas de dimensiones, se denomina **esquema en estrella**. Los diseños complejos de almacenes de datos pueden tener varios niveles de tablas de dimensiones; por ejemplo, la tabla *info-artículo* puede tener un atributo *id-fabricante* que es clave externa en otra tabla que da detalles del fabricante. Estos esquemas se denominan *esquemas en copo de nieve*. Los diseños complejos de almacenes de datos pueden tener también más de una tabla de hechos.

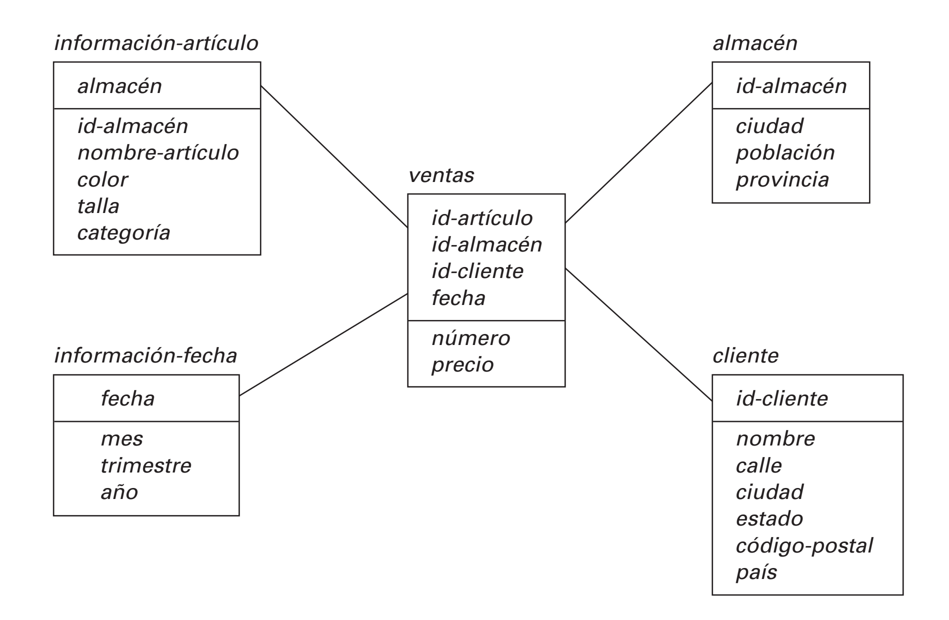


FIGURA 22.9. Esquema en estrella de un almacén de datos.

22.5. SISTEMAS DE RECUPERACIÓN DE LA INFORMACIÓN

El campo de la **recuperación de la información** se ha desarrollado en paralelo con el campo de las bases de datos. En el modelo tradicional utilizado en el campo de la recuperación de la información, la información se organiza en documentos, y se da por supuesto que hay un gran número de documentos. Los datos contenidos en los documentos están sin estructurar, sin ningún esquema asociado. El proceso de recuperación de la información consiste en localizar los documentos relevantes, en función de lo aportado por los usuarios, como pueden ser palabras clave o documentos de ejemplo.

La Web proporciona una manera adecuada de llegar a las fuentes de información y de interactuar con ellas a través de Internet. No obstante, un problema persistente que afronta la Web es la explosión de la información almacenada, con poca orientación para ayudar a los usuarios a localizar lo que es interesante. La recuperación de la información ha desempeñado un papel crítico en la transformación de la Web en una herramienta productiva y útil, especialmente para los investigadores.

Los ejemplos tradicionales de recuperación de la información son los catálogos en línea de las bibliotecas y los sistemas de gestión en línea de los documentos como los que almacenan los artículos de los periódicos. Los datos en esos sistemas están organizados como un conjunto de *documentos*; los artículos de un periódico o las entradas de un catálogo (en el catálogo de una biblioteca) son ejemplos de documentos. En el contexto de la Web, cada página HTML suele considerarse un documento.

Puede que el usuario de uno de estos sistemas desee recuperar un documento concreto o una clase de documentos determinada. Los documentos buscados suelen describirse mediante un conjunto de **palabras clave**; por ejemplo, las palabras clave «sistema de bases de datos» pueden utilizarse para localizar libros sobres sistemas de bases de datos, y las palabras clave «bolsa» y «escándalo» pueden utilizarse para localizar artículos sobre escándalos bursátiles. Los documentos tienen asociados conjuntos de palabras clave, y se recuperan los documentos cuyas palabras clave contienen las proporcionadas por el usuario.

La recuperación de información basada en las palabras clave no sólo se puede utilizar para recuperar datos textuales, sino también para recuperar otros tipos de datos, como los datos de vídeo o de audio, que tengan asociadas palabras clave descriptivas. Por ejemplo, una película de vídeo puede tener asociadas palabras clave como su título, su director, sus actores, el tipo de película, etcétera. Hay varias diferencias entre este modelo y los modelos utilizados en los sistemas tradicionales de bases de datos.

- Los sistemas de bases de datos tratan varias operaciones que no se abordan en los sistemas de recuperación de la información. Por ejemplo, los sistemas de bases de datos tratan con las actualizaciones y con los requisitos transaccionales asociados de control de la concurrencia y la durabilidad. Estos asuntos se consideran menos importantes en los sistemas de recuperación de la información. De mane-

ra parecida, los sistemas de bases de datos tratan con información estructurada organizada con modelos de datos relativamente complejos (como el modelo relacional o los modelos de datos orientados a los objetos), mientras que los sistemas de recuperación de la información han empleado tradicionalmente un modelo mucho más sencillo, en el que la información de la base de datos se organiza sencillamente como un conjunto de documentos sin estructurar.

- Los sistemas de recuperación de la información tratan varios aspectos que no se han abordado de manera adecuada en los sistemas de bases de datos. Por ejemplo, el campo de la recuperación de la información ha tratado los problemas de la gestión de documentos sin estructurar, como la búsqueda aproximada mediante palabras clave y la ordenación de los documentos por su grado estimado de importancia para la consulta.

22.5.1. Búsqueda por palabras clave

Los sistemas de recuperación de la información suelen permitir las expresiones de consulta formadas mediante palabras clave y las conectivas lógicas *y, o* y *no*. Por ejemplo, un usuario puede pedir todos los documentos que contengan las palabras clave «motocicleta *y* mantenimiento», o los documentos que contienen las palabras clave «computadora *o* microprocesador», o incluso los documentos que contienen la palabra clave «computadora *pero no* base de datos». Se da por supuesto que una consulta que contenga las palabras clave sin ninguno de las conectivas indicados tiene *y* conectando las palabras clave de manera implícita.

En la recuperación de **texto completo** se considera que todas las palabras de cada documento son palabras clave. Para los documentos no estructurados la recuperación de texto completo resulta fundamental, ya que puede que no haya información sobre las palabras del documento que son palabras clave. Se utilizará la palabra **término** para hacer referencia a las palabras de los documentos, ya que todas las palabras son palabras clave.

En su forma más sencilla los sistemas de recuperación de la información buscan y devuelven todos los documentos que contienen todas las palabras clave de la consulta, si es que no tiene conectivas; las conectivas se manejan de la forma esperada. Los sistemas más sofisticados estiman la importancia de los documentos para la consulta de modo que se puedan mostrar por orden de la importancia estimada. Estos sistemas utilizan información sobre las apariciones de los términos, así como la información de los hipervínculos para estimar la importancia; los apartados 22.5.1.1 y 22.5.1.2 esbozan el modo de hacerlo. El Apartado 22.5.1.3 esboza el modo de definir el parecido entre los documentos y su empleo para la búsqueda. Algunos sistemas también intentan ofrecer un conjunto mejor de respuestas utilizando el significado de los términos, en vez de limitarse a sus apariciones sintácticas, como se describe en el Apartado 22.5.1.4.

22.5.1.1. *Clasificación por la importancia mediante el empleo de términos*

El conjunto de los documentos que satisfacen una expresión de consulta puede ser muy grande; en concreto, hay miles de millones de documentos en la Web y la mayor parte de las consultas por palabras clave en los motores de búsqueda de la Web hallan centenares de millares de documentos que contienen esas palabras clave. La recuperación de texto completo agrava este problema: cada documento puede contener muchos términos, incluso términos que sólo se mencionan de pasada se tratan de manera equivalente a documentos en los que el término sí es importante. En consecuencia, puede que se recuperen documentos irrelevantes.

Por tanto, los sistemas de recuperación de la información estiman la importancia para la consulta de los documentos y sólo devuelven como respuestas los documentos con una clasificación más elevada. La clasificación por la importancia no es una ciencia exacta, pero hay algunos enfoques bien aceptados.

El primer punto que hay que abordar es, dado un término concreto t, la importancia para el término de un documento dado d. Un enfoque es utilizar el número de apariciones del término en el documento como medida de su importancia, con la suposición de que es probable que los términos importantes se mencionen muchas veces en el documento. El mero recuento del número de apariciones de un término no suele ser un buen indicador: en primer lugar, el número de apariciones depende de la longitud del documento y, en segundo lugar, puede que un documento que contenga diez apariciones de un término no tenga diez veces la importancia de un documento que contenga una sola aparición.

Un modo de medir $i(d, t)$, la importancia de un documento d para un término t, es

$$i(d, t) = \log\left(1 + \frac{n(d, t)}{n(d)}\right)$$

donde $n(d)$ denota el número de términos del documento y $n(d, t)$ denota el número de apariciones del término t en el documento d. Obsérvese que esta métrica tiene en cuenta la longitud del documento. La importancia crece con el número de apariciones del término en el documento, aunque no es directamente proporcional al número de apariciones.

Muchos sistemas refinan esta métrica empleando otra información. Por ejemplo, si el término aparece en el título, o en la lista de autores o en el resumen, el documento se considera más importante para el término. De manera parecida, si la primera aparición del término se produce muy avanzado el documento, puede que el documento se considere menos importante que si apa-

rece por primera vez al principio del documento. Las ideas anteriores pueden formalizarse mediante extensiones de la fórmula que se ha mostrado para $i(d, t)$. En la comunidad de recuperación de la información la importancia de un documento para un término se denomina **frecuencia del término**, independientemente de la fórmula concreta utilizada.

Una consulta C puede contener varias palabras clave. La importancia de un documento para una consulta con dos o más palabras clave se estima combinando las medidas de importancia del documento para cada palabra clave. Una manera sencilla de combinar las medidas es sumarlas. No obstante, no todos los términos utilizados como palabras clave son iguales. Supóngase que una consulta utiliza dos términos, uno de los cuales aparece con frecuencia, como «Web», y otro que es menos frecuente, como «Silberschatz». Un documento que contenga «Silberschatz» pero no «Web» debería clasificarse por encima de otro que contuviera «Web» pero no «Silberschatz».

Para solucionar este problema se asignan pesos a los términos empleando la **frecuencia inversa de los documentos**, definida como $1/n(t)$, donde $n(t)$ denota el número de documentos (entre los indexados por el sistema) que contienen el término t. La **importancia** de un documento d para un conjunto de términos Q se define como

$$i(d,Q) = \sum_{t \in Q} \frac{r(d, t)}{n(t)}$$

Esta medida puede refinarse aún más si se permite a los usuarios especificar los pesos $p(t)$ de los términos de la consulta, en cuyo caso los pesos especificados por los usuarios se tienen también en cuenta empleando $p(t)/n(t)$ en lugar de $1/n(t)$.

Casi todos los documentos de texto (en español) contienen palabras como «y», «o», «un», etcétera y, por tanto, estas palabras resultan inútiles para propósitos de consulta, ya que su frecuencia inversa de documentos es extremadamente baja. Los sistemas de recuperación de la información definen un conjunto de palabras, denominadas **palabras de parada**, que contienen aproximadamente cien de las palabras más frecuentes, y eliminan este conjunto del documento al indexarlo; estas palabras no se utilizan como palabras clave, y se descartan si se hallan entre las palabras proporcionadas por los usuarios.

Otro factor que se tiene en cuenta cuando una consulta contiene varios términos es la **proximidad** de los términos en el documento. Si los términos aparecen cercanos entre sí en el documento, el documento se clasificará en una posición más elevada que si aparecen muy separados. La fórmula de $i(d,Q)$ puede modificarse para tener en cuenta la proximidad.

Dada una consulta C, el trabajo del sistema de recuperación de la información es devolver documentos en orden descendente de importancia para C. Dado que puede haber un gran número de documentos que carezcan de importancia, los sistemas de recuperación de la información suelen devolver únicamente los primeros documentos con el grado más elevado de importancia estimada y permiten a los usuarios solicitar de manera interactiva más documentos.

22.5.1.2. La importancia cuando se utilizan hipervínculos

Los primeros motores de búsqueda de la Web clasificaban los documentos utilizando sólo medidas de importancia parecidas a las descritas en el Apartado 22.5.1.1. Sin embargo, los investigadores pronto se dieron cuenta de que los documentos Web contienen información de la que carecen los documentos de texto sencillo, por ejemplo, los hipervínculos. Y, de hecho, la ordenación por la importancia de los documentos se ve más afectada por los hipervínculos que apuntan *al* documento que por los hipervínculos que proceden del documento.

La idea básica de la clasificación de los sitios es hallar sitios que sean populares, y clasificar las páginas de esos sitios por encima de las páginas de otros sitios. Un sitio queda identificado por la parte de la dirección de internet de su URL, como, por ejemplo, www.bell-labs.com en la URL http://www.bell-labs.com/topic/books/db-book. Cada sitio suele contener varias páginas web. Dado que la mayor parte de las búsquedas pretenden hallar información de los sitios populares, la clasificación de las páginas de los sitios populares por encima de las demás suele ser una buena idea. Por ejemplo, el término «google» puede aparecer en gran número de páginas, pero el sitio google.com es el sitio más popular de entre los sitios con páginas que contienen el término «google». Los documentos procedentes de google.com que contengan el término «google» se clasificarán, por tanto, como los más importantes para el término «google».

Esto suscita la pregunta del modo de definir la popularidad de un sitio. Una manera es hallar la cantidad de veces que se tiene acceso a ese sitio. Sin embargo, la obtención de esa información es imposible sin la cooperación del sitio, y no les es factible implementarla a los motores de búsqueda web. Una alternativa muy efectiva utiliza los hipervínculos; define $p(s)$, la **popularidad del sitio** s, como el número de sitios que contienen como mínimo una página con un vínculo con el sitio s.

Las medidas tradicionales de la importancia de una página (que se vieron en el Apartado 22.5.1.2) pueden combinarse con la popularidad del sitio que contiene la página para obtener una medida global de la importancia de la página. Las páginas con un valor elevado de su importancia global se devuelven como respuestas de la consulta, como ya se ha visto.

También hay que tener en cuenta que se ha utilizado la popularidad del *sitio* como medida de la importancia de las páginas de ese sitio, no la popularidad de cada una de las *páginas*. Hay, como mínimo, dos razones para ello. En primer lugar, la mayor parte de los sitios sólo contienen páginas a las páginas iniciales de

los otros sitios, por lo que parecería que todas las demás páginas tienen una popularidad casi nula, cuando en realidad puede que se tenga acceso bastante frecuente a ellas siguiendo los vínculos desde la página inicial. En segundo lugar, hay muchos menos sitios que páginas, por lo que el cálculo y el empleo de la popularidad de los sitios resulta más económico que el cálculo y el empleo de la popularidad de las páginas.

Hay definiciones más refinadas de la popularidad de los sitios. Por ejemplo, puede que se considere un vínculo desde un sitio popular a otro sitio *s* una indicación mejor de la popularidad de *s* que un vínculo con *s* desde un sitio menos popular[6]. Esta definición de popularidad es, de hecho, circular, ya que la popularidad de un sitio queda definida por la popularidad de otros sitios y puede que haya ciclos de vínculos entre los sitios. No obstante, la popularidad de los sitios puede definirse mediante un sistema de ecuaciones lineales simultáneas, que pueden resolverse mediante las técnicas de tratamiento de matrices. Las ecuaciones lineales se definen de manera que tienen una solución única y bien definida.

El popular motor de búsqueda Web google.com utiliza la idea de popularidad de los sitios remitentes en su definición de **clasificación de las páginas**, que es una medida de la popularidad de cada página. Este enfoque de la clasificación de las páginas ha dado muchos mejores resultados que las técnicas de clasificación utilizadas anteriormente, por lo que google.com se ha convertido en un motor de búsqueda muy utilizado en un periodo de tiempo bastante breve.

Hay otro enfoque, en cierto modo parecido, que, curiosamente, procede de una teoría de redes sociales desarrollada por los sociólogos en los años cincuenta del siglo pasado. En el contexto de las redes sociales el objetivo era definir el prestigio de las personas. Por ejemplo, el presidente de los Estados Unidos tiene un elevado prestigio, ya que gran cantidad de gente lo conoce. Si alguien es conocido por varias personas de prestigio, también tendrá un prestigio elevado, aunque no sea conocido por un número de personas tan grande.

La idea anterior se desarrolló hasta una definición de *nodos* y de *autoridades* que tiene en cuenta la presencia de directorios para enlazar con páginas que contienen información útil. Un **nodo** es una página que almacena vínculos con muchas páginas; no contiene información real sobre ningún asunto, pero apunta a páginas que sí que la contienen. Por el contrario, una **autoridad** es una página que contiene información real sobre un asunto, aunque puede que no apunten a ella muchas páginas. Cada página recibe un valor de prestigio como nodo (*prestigio-nodo*) y otro valor de prestigio como autoridad (*prestigio-autoridad*). Las definiciones de prestigio, como ya se ha visto, son cíclicas y vienen dadas por un conjunto de ecuaciones lineales simultáneas. Una página obtiene un mayor prestigio-nodo si apunta a muchas páginas con un elevado prestigio-autoridad, mientras que una página recibe un elevado prestigio-autoridad si apuntan a ella muchas páginas con un elevado prestigio-nodo. Dada una consulta, las páginas con prestigio-autoridad elevado se clasifican por encima de las demás páginas. Véanse las notas bibliográficas para hallar referencias que ofrezcan más detalles.

22.5.1.3. *Recuperación basada en la semejanza*

Algunos sistemas de recuperación de la información permiten la **recuperación basada en la semejanza**. En este caso, el usuario puede dar al sistema el documento *A* y pedirle que recupere documentos que sean «semejantes» a *A*. El parecido de un documento con otro puede definirse, por ejemplo, con base en los términos comunes. Un enfoque es hallar *k* términos de *A* con los valores más elevados de $i(d, t)$ y emplear esos *k* términos como consulta para hallar la importancia de otros documentos. Los términos de la consulta se pesan con $i(d, t)$.

Si el conjunto de documentos semejantes a *A* es de gran tamaño, puede que el sistema sólo presente al usuario unos cuantos documentos, le permita escoger los más destacados e inicie una nueva búsqueda basada en el parecido con *A y* con los documentos seleccionados. Es posible que el conjunto de documentos resultante sea lo que el usuario pretendía hallar.

La misma idea se utiliza también para ayudar a los usuarios que hallan muchos documentos que parecen ser importantes de acuerdo con las palabras clave pero que no lo son. En esas situaciones, en lugar de añadir más palabras clave a la consulta, puede que se permita a los usuarios identificar uno o varios de los documentos devueltos como importantes; el sistema utilizará los documentos identificados para hallar otros parecidos. Es probable que el conjunto de documentos resultante sea lo que el usuario pretendía hallar.

22.5.1.4. *Sinónimos y homónimos*

Considérese el problema de la localización de documentos sobre el mantenimiento de motocicletas con las palabras clave «motocicleta» y «mantenimiento». Supóngase que las palabras clave para cada documento sean las palabras del título y los nombres de los autores. El documento titulado *Reparación de motocicletas* no se recuperaría, ya que la palabra «mantenimiento» no aparece en el título.

Se puede resolver el problema haciendo uso de los **sinónimos**. Cada palabra puede tener definido un con-

[6] Esto es parecido, en cierto sentido, a conceder un peso extra al aval de productos por gente famosa (como las estrellas de cine), por lo que su importancia es discutible.

junto de sinónimos, y la aparición de una palabra puede ser sustituida por la *disyunción* de todos sus sinónimos (incluida la propia palabra). Así, la consulta «motocicleta *y* reparación» puede sustituirse por «motocicleta *y* (reparación *o* mantenimiento)». Esta consulta sí hallaría el documento deseado.

Las consultas basadas en las palabras clave también se ven afectadas por el problema, el de los **homónimos**, es decir, las palabras con varios significados. Por ejemplo, la palabra objeto tiene significados diferentes como nombre y como verbo. La palabra tabla puede hacer referencia a una pieza de madera o a una tabla relacional. Algunos sistemas de consultas por palabras clave intentan deshacer la ambigüedad del significado de las palabras en los documentos, y cuando un usuario formula una consulta averiguan el significado deseado preguntándoselo. Los documentos devueltos son los que utilizan el término con el significado deseado por el usuario. Sin embargo, la ruptura de la ambigüedad de los significados de las palabras de los documentos no resulta una tarea sencilla, por lo que no muchos sistemas implementan esta idea.

De hecho, un riesgo de utilizar los sinónimos para ampliar las consultas es que los sinónimos pueden tener significados diferentes. Se recuperarán documentos que utilizan los sinónimos con un significado implícito alternativo y el usuario se pregunta el motivo de que el sistema considerara que uno de los documentos recuperados era importante, si no contiene ni las palabras clave especificadas por el usuario ni las palabras cuyo significado implícito en el documento es sinónimo de las palabras clave especificadas. Por tanto, es recomendable comprobar los sinónimos con el usuario antes de utilizarlos para ampliar una consulta que éste haya formulado.

22.5.2. Indexado de documentos

Una estructura efectiva de índices es importante para el procesamiento eficiente de las consultas en los sistemas de recuperación de la información. Los documentos que contengan las palabras clave especificadas pueden localizarse de manera efectiva utilizando un **índice invertido**, que relaciona cada palabra clave K_i con el conjunto S_i de (los identificadores de) los documentos que contienen K_i. Para dar soporte a la clasificación de importancia basada en la proximidad de las palabras clave estos índices pueden proporcionar no sólo los identificadores de los documentos, sino también una lista de las ubicaciones en el documento en las que aparecen las palabras clave.

Dado que esos índices hay que almacenarlos en disco, la organización del índice también intenta minimizar el número de operaciones de E/S para la recuperación del conjunto de (identificadores de) documentos que contienen las palabras clave. Así, puede que el sistema intente guardar el conjunto de documentos de cada palabra clave en páginas consecutivas del disco.

La operación *y* halla los documentos que contienen todas las palabras clave de un conjunto especificado $K_1, K_2, \ldots, K_n$. La operación *y* se implementa recuperando primero los conjuntos de identificadores de documentos $S_1, S_2, \ldots, S_n$ de todos los documentos que contienen las palabras clave respectivas.

La intersección $S_1 \cap S_2 \cap \cdots \cap S_n$, de los conjuntos de documentos da los identificadores de los documentos del conjunto de documentos deseado. La operación *o* da el conjunto de todos los documentos que contienen al menos una de las palabras clave $K_1, K_2, \ldots, K_n$. La operación *o* se implementa calculando la unión, $S_1 \cup S_2 \cup \cdots \cup S_n$, de los conjuntos. La operación *no* halla los documentos que no contienen la palabra clave especificada K_i. Dado un conjunto de identificadores S, se pueden eliminar los documentos que contengan la palabra clave especificada K_i tomando la diferencia $S - S_i$, donde S_i es el conjunto de identificadores de los documentos que contienen la palabra clave K_i.

Dado un conjunto de palabras clave de una consulta, muchos sistemas de recuperación de la información no insisten en que los documentos recuperados contengan todas las palabras clave (a menos que se utilice de manera explícita una operación *y*). En ese caso, se recuperan todos los documentos que contengan como mínimo una de las palabras clave (como en la operación *o*), pero se clasifican de acuerdo con su medida de importancia.

Para utilizar en la clasificación la frecuencia de los términos, la estructura de índices también debe mantener el número de veces que aparecen los términos en cada documento. Para reducir este esfuerzo pueden utilizar una representación comprimida con sólo unos pocos bits, que aproxima la frecuencia de los términos. El índice también debe almacenar la frecuencia de documentos de cada término (es decir, el número de documentos en que aparece cada término).

22.5.3. Medida de la efectividad de la recuperación

Cada palabra clave puede estar incluida en gran número de documentos; por tanto, una representación compacta resulta fundamental para mantener bajas las necesidades de espacio del índice. Así, los conjuntos de documentos de cada palabra clave se conservan en forma comprimida. Para ahorrar espacio de almacenamiento a veces se almacena el índice de modo que la recuperación es aproximada; puede que no se recuperen unos pocos documentos de importancia (lo que se denomina un **rechazo falso** o un **falso negativo**), o puede que se recuperen unos pocos documentos sin importancia (lo que se denomina un **falso positivo**). Una buena estructura de índice no tiene *ningún* rechazo falso, pero puede que permita algunos falsos positivos; el sistema puede filtrarlos posteriormente mirando las palabras clave que contienen realmente. En el indexado de Webs los falsos positivos tampoco son deseables, ya

que puede que el documento real no sea accesible rápidamente para su filtrado.

Se utilizan dos métricas para medir la calidad con que los sistemas de recuperación de la información pueden contestar las consultas. La primera, la **precisión**, mide el porcentaje de los documentos recuperados que son verdaderamente importantes para la consulta. La segunda, la **recuperación**, mide el porcentaje de los documentos importantes para la consulta que se han recuperado. Lo ideal sería que ambas fueran del ciento por ciento.

La precisión y la recuperación también son medidas importantes para la comprensión de la calidad de una determinada estrategia de clasificación de los documentos. Las estrategias de clasificación pueden dar lugar a falsos negativos y en falsos positivos, pero en un sentido más sutil.

- Pueden producirse falsos negativos al clasificar los documentos porque los documentos importantes obtengan clasificaciones bajas; si se extrajeran todos los documentos hasta incluir los que tienen clasificaciones muy bajas, habría muy pocos falsos negativos. Sin embargo, los usuarios rara vez miran más allá de las primeras decenas de documentos devueltos y, por tanto, puede que pasen por alto documentos importantes porque no estén clasificados entre los primeros. Lo que sea exactamente un falso negativo depende del número de documentos que se examinen.

  Por tanto, en lugar de tener un solo número como medida de la recuperación, se puede medir la recuperación como una función del número de documentos extraídos.
- Pueden producirse falsos positivos porque documentos sin importancia obtengan clasificaciones más elevadas que algunos documentos importantes. Esto también depende del número de documentos que se examinen. Una posibilidad es medir la precisión como función del número de documentos extraídos.

Una opción mejor y más intuitiva para la medida de la precisión es medirla como función de la recuperación. Con esta medida combinada, tanto la precisión como la recuperación pueden calcularse en función del número de documentos, si hiciera falta.

Por ejemplo, se puede decir que con una recuperación del 50 por ciento la precisión era del 75 por ciento, mientras que con una recuperación del 75 por ciento la precisión cayó hasta el 60 por ciento. En general, se puede dibujar una gráfica que relacione la precisión con la recuperación. Estas medidas pueden calcularse para las diferentes consultas y promediarse para un conjunto de consultas en una prueba de homologación de la calidad de las consultas.

Otro problema más con la medida de la precisión y de la recuperación consiste en el modo de definir los documentos que son realmente importantes y los que no lo son. De hecho, decidir si un documento es importante o no lo es exige la comprensión del lenguaje natural y de las pretensiones de la consulta. Los investigadores, por tanto, han creado conjuntos de documentos y de consultas y han marcado manualmente los documentos como importantes o no importantes para las consultas. Se pueden ejecutar diferentes sistemas de clasificación con estos conjuntos de documentos para medir la precisión y la recuperación medias de varias consultas.

22.5.4. Motores de búsqueda web

Los programas denominados **web crawlers** (batidores web) son programas que localizan y reúnen información de la Web. Siguen de manera recursiva los hipervínculos presentes en los documentos conocidos para hallar otros documentos. Los batidores recuperan los documentos y añaden la información hallada en ellos a índices combinados; generalmente, los documentos no se almacenan, aunque algunos motores de búsqueda guardan en la caché una copia del documento para dar a los clientes un acceso más rápido a los documentos. Dado que el número de documentos de la Web es muy grande, no es posible recorrer toda la Web en un periodo corto de tiempo; y, de hecho, todos los motores de búsqueda cubren únicamente algunas partes de la Web, no toda ella, y sus batidores pueden tardar semanas o meses en realizar una sola batida de todas las páginas que abarcan. Suele haber muchos procesos ejecutándose en varias máquinas, implicadas en las batidas. Una base de datos almacena un conjunto de vínculos (o de sitios) que hay que batir; asigna los vínculos que parten de este conjunto a cada proceso batidor. Los vínculos nuevos hallados durante cada batida se añaden a la base de datos, y puede que se batan posteriormente si no se baten de manera inmediata. Las páginas halladas durante cada batida también se pasan al sistema de indexado, que puede que se ejecute en una máquina diferente. Hay que volver a extraer las páginas (es decir, volver a batir los vínculos) de manera periódica para obtener información actualizada y descartar los sitios que ya no existan, de modo que la información del índice de búsqueda se mantenga razonablemente actualizada.

El propio sistema de indexado se ejecuta en paralelo en varias máquinas. No resulta una buena idea añadir las páginas al mismo índice que se utiliza para las consultas, ya que eso exige el control de la concurrencia en el índice y afecta al rendimiento de las consultas y de las actualizaciones. En lugar de eso, se utiliza una copia del índice para responder a las consultas mientras se actualiza otra copia con las páginas recién batidas. A intervalos periódicos se intercambian las copias y se actualiza la más antigua mientras la copia nueva se utiliza para las consultas.

Para soportar tasas de consultas muy elevadas se pueden mantener los índices en la memoria principal y, si

hay varias máquinas, el sistema encamina de manera selectiva las consultas a las máquinas para equilibrar la carga de trabajo entre ellas.

22.5.5. Directorios

El usuario típico de una biblioteca puede utilizar un catálogo para hallar el libro que busca. Cuando recupera el libro del estante, no obstante, es probable que *hojee* otros libros que se hallen cerca. Las bibliotecas organizan los libros de modo que los títulos relacionados se guarden cerca unos de otros. Por tanto, puede que un libro que esté físicamente cerca del libro deseado también sea interesante, lo que hace interesante para los usuarios hojear esos libros.

Para guardar cerca unos de otros los libros relacionados, las bibliotecas utilizan una **jerarquía de clasificación**. Los libros de ciencias se clasifican juntos. Dentro de este conjunto de libros hay una clasificación más fina, que organiza por un lado los libros de informática, por otro los de matemáticas, etcétera. Dado que hay una relación entre las matemáticas y la informática, hay conjuntos importantes de libros que se guardan físicamente cerca. En otro nivel diferente de la jerarquía de la clasificación, los libros de informática se dividen en subáreas, como los sistemas operativos, los lenguajes y los algoritmos. La Figura 22.10 muestra una jerarquía de clasificación que las bibliotecas pueden emplear. Como los libros sólo se pueden guardar en un sitio, cada libro de una biblioteca se clasifica exactamente en un punto de la jerarquía de clasificación.

En los sistemas de recuperación de la información no hace falta almacenar cerca los documentos relacionados. No obstante, estos sistemas necesitan *organizar lógicamente los documentos* para permitir su exploración. Por tanto, estos sistemas pueden utilizar una jerarquía de clasificación parecida a la que utilizan las bibliotecas y, cuando muestra un documento concreto, también puede mostrar una breve descripción de los documentos que se hallan cercanos en la jerarquía.

En los sistemas de recuperación de la información no hace falta guardar cada documento en un solo punto de la jerarquía. Los documentos que traten de matemáticas para informáticos pueden clasificarse en matemáticas y en informática. Todo lo que se guarda en cada punto es un identificador del documento (es decir, un puntero hacia el documento), y resulta sencillo extraer el contenido del documento empleando el identificador.

Como consecuencia de esa flexibilidad, no sólo se puede clasificar cada documento en dos posiciones, sino que también puede aparecer una subárea de la jerarquía de la clasificación en dos áreas diferentes. La clase de documento «algoritmo de grafos» puede aparecer tanto en matemáticas como en informática. Por tanto, la jerarquía de clasificación es ahora un grafo acíclico dirigido (GAD), como puede verse en la Figura 22.11. Los documentos de algoritmos de grafos pueden aparecer en una sola posición del GAD, pero puede llegarse a ellos por varios caminos.

Un **directorio** no es más que una estructura de clasificación GAD. Cada hoja del directorio almacena los vínculos con documentos del tema representado por la hoja. Los nodos internos también pueden contener vínculos, por ejemplo, con documentos que no se pueden clasificar en ninguno de los nodos hijo.

Para hallar información sobre un asunto los usuarios empiezan en la raíz del directorio y siguen los caminos por el GAD hasta alcanzar el nodo que representa el asunto deseado. Mientras avanzan por el directorio los usuarios no sólo pueden hallar documentos sobre el asunto en el que están interesados, sino que también

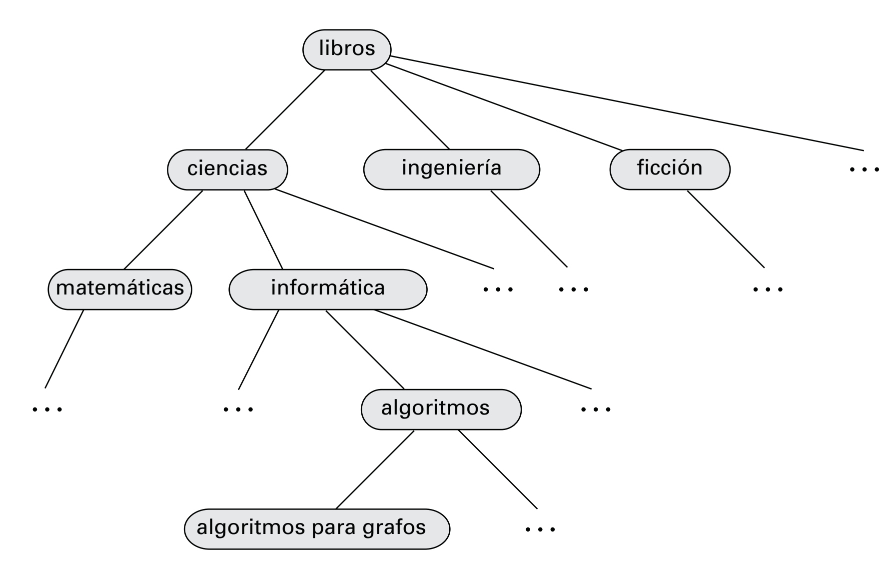


FIGURA 22.10. Jerarquía de clasificación para el sistema de una biblioteca.

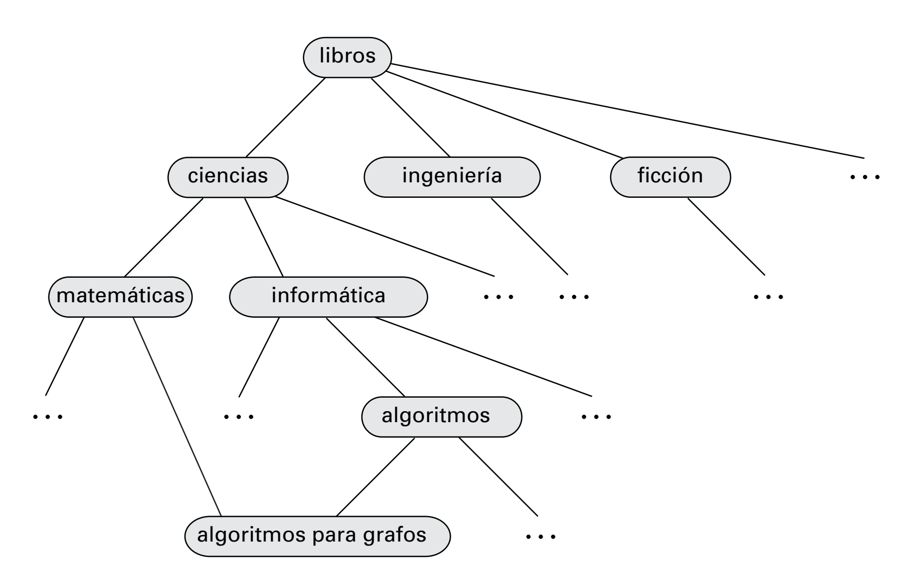


FIGURA 22.11. GAD de calsificación del sistema de recuperación de información de una biblioteca.

pueden hallar documentos relacionados y clases relacionadas en la jerarquía de clasificación. Los usuarios pueden conseguir información nueva explorando los documentos (o las subclases) de las clases relacionadas.

La organización de la enorme cantidad de información disponible en la Web en una estructura de directorio es una tarea aterradora.

- El primer problema es la determinación de cuál debe ser exactamente la jerarquía del directorio.
- El segundo problema es, dado un documento, decidir los nodos del directorio que son categorías importantes para el documento.

Para afrontar el primer problema, los portales como Yahoo tienen equipos de «bibliotecarios de Internet» que sugieren la jerarquía de la clasificación y la refinan de manera continua. El *Proyecto de directorio abierto* (*Open Directory Project*) es un gran esfuerzo cooperativo con diferentes voluntarios que son responsables de organizar diferentes ramas del directorio.

El segundo problema también pueden afrontarlo manualmente los bibliotecarios, o puede que los conservadores del sitio web sean responsables de decidir el lugar de la jerarquía en que deben ubicarse sus sitios. También hay técnicas para decidir de manera automática la ubicación de los documentos de acuerdo con el cálculo de su parecido con documentos que ya se hayan clasificado.

22.6. RESUMEN

- Los sistemas de ayuda a la toma de decisiones analizan en línea los datos recogidos por los sistemas de procesamiento de transacciones para ayudar a los usuarios a adoptar decisiones de negocios. Dado que hoy en día la mayor parte de las organizaciones están intensamente informatizadas, se dispone de una enorme cantidad de información para la ayuda a la toma de decisiones. Los sistemas de ayuda a la toma de decisiones se presentan en varios formatos, incluidos los sistemas OLAP y los sistemas de recopilación de datos.
- Las herramientas de procesamiento analítico en línea (online analytical processing, OLAP) ayudan a los analistas a ver los datos resumidos de diferentes maneras, de manera que puedan obtener una perspectiva del funcionamiento de la organización.

— Las herramientas OLAP trabajan con datos multidimensionales, caracterizados por los atributos de dimensiones y por los atributos de medida.

— Los cubos de datos consisten en datos multidimensionales resumidos de diferentes maneras. El cálculo previo de los cubos ayuda a acelerar las consultas de resúmenes de datos.

— El formato de las tabulaciones cruzadas permite que los usuarios vean simultáneamente dos dimensiones de los datos multidimensionales, junto con resúmenes de los datos.

— La concreción, la abstracción y los cortes de los cubos de datos son algunas de las operaciones que los usuarios llevan a cabo con las herramientas OLAP.

- El componente OLAP de la norma SQL:1999 proporciona gran variedad de funcionalidades nuevas para el análisis de los datos, incluidas nuevas funciones de agregación, operaciones con cubos y de abstracción, funciones de clasificación, funciones para la creación de ventanas, que soportan la elaboración de resúmenes en ventanas móviles, y la realización de particiones, con la creación de ventanas y la clasificación aplicadas dentro de cada partición.
- La recuperación de datos es el proceso de análisis semiautomático de bases de datos de gran tamaño para hallar estructuras útiles. Hay gran número de aplicaciones de recopilación de datos, como la predicción de valores con base en los ejemplos ya pasados, la búsqueda de asociaciones entre las compras y la agrupación automática de personas y películas.
- La clasificación trata de la predicción de la clase de los ejemplos de prueba utilizando los atributos de los ejemplos de prueba con base en los atributos de los ejemplos de formación y en la clase real de los ejemplos de formación. La clasificación puede utilizarse, por ejemplo, para predecir el valor de crédito de los nuevos solicitantes o para predecir el rendimiento de los estudiantes que solicitan el ingreso en una universidad.

  Hay varios tipos de clasificadores, como los

— Clasificadores de árboles de decisión. Estos llevan a cabo la clasificación creando un árbol basado en los ejemplos de formación con hojas que tienen las etiquetas de las clases. Se recorre el árbol para cada ejemplo de prueba hasta hallar una hoja y la clase de esa hoja es la clase predicha.

— Se dispone de varias técnicas para crear árboles de decisión, la mayor parte de ellas basadas en heurística codiciosa.

- Los clasificadores bayesianos son más sencillos de crear que los clasificadores de árboles de decisión y funcionan mejor en caso de que haya valores de los atributos que falten o que sean nulos.
- Las reglas de asociación identifican los elementos que aparecen juntos con frecuencia, por ejemplo, los artículos que el mismo cliente suele comprar. Las correlaciones buscan las desviaciones respecto de los niveles de asociación esperados.
- Otros tipos de recopilación de datos son el agrupamiento, la recopilación de texto y la visualización de datos.
- Los almacenes de datos ayudan a reunir y archivar los datos operativos importantes. Los almacenes de datos se utilizan para la ayuda a la toma de decisiones y para el análisis de datos históricos, por ejemplo, para predecir tendencias. La limpieza de los datos de los orígenes de datos de entrada suele ser una tarea importante en el almacenaje de datos. Los esquemas de los almacenes de datos suelen ser multidimensionales e implican una o pocas tablas de hechos muy grandes y varias tablas de dimensiones mucho más pequeñas.
- Los sistemas de recuperación de la información se utilizan para almacenar y consultar los datos de texto como los documentos. Utilizan un modelo de datos más sencillo que el de los sistemas de bases de datos, pero ofrecen posibilidades de búsqueda más potentes dentro de ese modelo restringido.

  Las consultas intentan localizar los documentos de interés especificando, por ejemplo, conjuntos de palabras clave. La consulta que el usuario tiene en mente no suele poder formularse de manera precisa; por tanto, los sistemas de recuperación de la información ordenan las respuestas con base en su importancia potencial.
- La clasificación por importancia hace uso de varios tipos de información como:

— Frecuencia de los términos: la importancia de cada término para cada documento.

— Frecuencia inversa de los documentos.

— Popularidad del sitio. La clasificación de las páginas y la clasificación del nodo o de la autoridad son dos maneras de asignar importancia a los sitios de acuerdo con los vínculos con cada sitio.

- La semejanza de los documentos se utiliza para recuperar documentos parecidos a un documento de ejemplo. Los sinónimos y los homónimos complican la tarea de la recuperación de la información.
- La precisión y la recuperación son dos medidas de la efectividad de los sistemas de recuperación de la información.
- Las estructuras de directorio se utilizan para clasificar los documentos con otros documentos parecidos.

TÉRMINOS DE REPASO

- Agregación ampliada
  — Correlación
  — Desviación estándar
  — Regresión
  — Varianza
- Agrupamiento
  — Agrupamiento aglomerativo
  — Agrupamiento divisivo
  — Agrupamiento jerárquico
- Almacenamiento de datos

- — Arquitectura dirigida por el destino
- — Arquitectura dirigida por el origen
- — Limpieza de datos
  - – Domiciliación
  - – Mezcla-purga
- — Recogida de datos
- Análisis estadístico
- Asociaciones
- Batidores web
- Búsqueda por palabras clave
- Clasificación
  - — Datos de formación
  - — Datos de prueba
- Clasificación por importancia
  - — Frecuencia inversa de los documentos
  - — Frecuencia de los términos
  - — Importancia
  - — Proximidad
- Clasificadores de árboles de decisión
  - — Atributo categórico
  - — Atributo de partición
  - — Atributo con valores continuos
  - — Condición de partición
  - — Contenido de la información
  - — División binaria
  - — División múltiple
  - — Exceso de ajuste
  - — Ganancia de información
  - — Pureza
    - – Medida de la entropía
    - – Medida de Gini
  - — Tasa de ganancia de la información
- Clasificadores bayesianos
  - — Clasificadores bayesianos ingenuos
  - — Teorema de Bayes
- Creación de ventanas
- Cubo de datos
- Datos multidimensionales
  - — Atributos de dimensiones
  - — Atributos de medida
- Directorios
- Esquemas de almacenamiento
  - — Esquema en estrella
  - — Tablas de dimensiones
  - — Tabla de hechos
- Falso negativo
- Falso positivo
- Funciones de clasificación
  - — Clasificación
  - — Clasificación densa
  - — División mediante
- Homónimos
- Importancia cuando se utilizan hipervínculos
  - — Clasificación nodo/autoridad
  - — Clasificación de las páginas
  - — Popularidad de un sitio
- Índice invertido
- Jerarquía de clasificación
- OLAP híbrido (HOLAP)
- OLAP multidimensional (MOLAP)
- OLAP relacional (ROLAP)
- Otros tipos de asociaciones
- Palabras de parada
- Precisión
- Predicción
- Procesamiento analítico en línea (online analytical processing, OLAP)
  - — Abstracción y concreción
  - — Cortes de cubos
  - — Pivotaje
- Rechazo falso
- Recopilación de datos
- Recopilación de texto
- Recuperación
- Recuperación basada en la semejanza
- Recuperación de texto completo
- Reglas de asociación
  - — Confianza
  - — Conjuntos de elementos de gran tamaño
  - — Población
  - — Soporte
- Regresión
  - — Ajuste de curvas
  - — Regresión lineal
- Sinónimos
- Sistemas de ayuda a la toma de decisiones
- Sistemas de recuperación de la información
- Tabulaciones cruzadas
- Término
- Visualización de datos

EJERCICIOS

22.1. Para cada una de las funciones de agregación de SQL **sum, count, min** y **max**, muéstrese el modo de calcular el valor agregado para el conjunto múltiple $S_1 \cup S_2$, dados los valores agregados para los conjuntos múltiples S_1 y S_2.

Con base en lo anterior hay que dar las expresiones para calcular los valores agregados con agrupamiento en para un subconjunto S de los atributos de la relación $r(A,B,C,D,E)$, dados los valores agregados para agrupamiento de los atributos $T \supseteq S$, para las siguientes funciones de agregación:

a. **sum, count, min** y **max**

b. **avg**

c. Desviación estándar

22.2. Muéstrese el modo de expresar **group by cube**(*a, b, c, d*) utilizando **rollup**; la respuesta sólo debe tener una cláusula **group by**.

22.3. Dese un ejemplo de un par de agrupamientos que no puedan expresarse utilizando una sola cláusula **group by** con **cube** y **rollup**.

22.4. Dada la relación *E*(*estudiante, asignatura, notas*), escríbase una consulta para hallar los *n* mejores estudiantes por sus notas totales, utilizando la clasificación.

22.5. Dada la relación *r*(*a, b, d, d*), muéstrese el modo de utilizar las características ampliadas de SQL para generar un histograma de *d* frente a *a*, dividiendo *a* en veinte particiones de igual tamaño (es decir, que cada partición contiene el 5 por ciento de las tuplas de *r*, ordenadas según *a*).

22.6. Escríbase una consulta para hallar saldos acumulativos, equivalente a la mostrada en el Apartado 22.2.5, pero sin utilizar las estructuras ampliadas para la creación de ventanas de SQL.

22.7. Considérese el atributo *saldo* de la relación *cuenta*. Escríbase una consulta en SQL para calcular un histograma de los valores de *saldo*, dividiendo el rango desde cero hasta el máximo saldo de una cuenta presente, en tres rangos iguales.

22.8. Considérese la relación *ventas* del Apartado 22.2. Escríbase una consulta en SQL para calcular la operación cubo para la relación, dada la relación de la Figura 22.2. No hay que utilizar la estructura **with cube**.

22.9. Créese un clasificador de árboles de decisión con divisiones binarias en cada nodo utilizando las tuplas de la relación *r*(*A,B,C*) que se muestra más abajo como datos de formación; el atributo *C* denota la clase. Muéstrese el árbol final y, con cada nodo, muéstrese la mejor división para cada atributo junto con su valor de ganancia de la información.

(1, 2, *a*), (2, 1, *a*), (2, 5, *b*), (3, 3, *b*), (3, 6, *b*),
(4, 5, *b*), (5, 5, *c*), (6, 3, *b*), (6, 7, *c*)

22.10. Supóngase que hay dos reglas de clasificación, una que dice que la gente con sueldos entre 10.000 € y 20.000 € tienen una calificación de crédito de *buena*, y otra que dice que la gente con sueldos entre 20.000 € y 30.000 € tienen una calificación de crédito de *buena*. Hay que indicar las condiciones para las que se pueden sustituir las reglas, sin pérdida de información, por una sola regla que diga que las personas con sueldos entre 10.000 € y 30.000 € tienen una calificación de crédito de *buena*.

22.11. Supóngase que la mitad de las transacciones de una tienda de ropa adquieren vaqueros y un tercio de las transacciones de la tienda adquieren camisetas. Supóngase también que la mitad de las transacciones que adquieren vaqueros también adquieren camisetas. Hay que anotar todas las reglas de asociación (no triviales) que se puedan deducir de esta información, dando el soporte y la confianza de cada regla.

22.12. Considérese el problema de hallar conjuntos de artículos de gran tamaño.

a. Descríbase el modo de hallar el soporte de un conjunto dado de conjuntos de elementos mediante una sola exploración de los datos. Supóngase que los conjuntos de artículos y la información asociada, como los recuentos, caben en la memoria.

b. Supóngase que un conjunto de artículos tiene un soporte menor que *j*. Muéstrese que ningún superconjunto de este conjunto de artículos puede tener un soporte mayor o igual que *j*.

22.13. Descríbanse las ventajas y los inconvenientes de una arquitectura dirigida por el origen para la recolección de datos en los almacenes de datos en comparación con una arquitectura dirigida por el destino.

22.14. Considérese el esquema dibujado en la Figura 22.9. Dada una consulta de SQL:1999 para resumir las cifras de ventas y los precios por tienda y por fecha, junto con las jerarquías para tienda y fecha.

22.15. Calcúlese la importancia (mediante las definiciones adecuadas de la frecuencia de los términos y de la frecuencia inversa de los documentos) de cada una de las preguntas de este capítulo para la consulta «relación SQL».

22.16. Explíquese la diferencia entre un falso positivo y un rechazo falso. Si es fundamental que las consultas de recuperación de la información no pierdan ninguna información importante, explicar si es aceptable tener falsos positivos o rechazos falsos. Explicar el motivo.

22.17. Supóngase que se desea hallar documentos que contengan como mínimo *k* palabras clave de un conjunto dado de *n*. Supóngase también que se dispone de un índice de palabras clave que da una lista (ordenada) de identificadores de documentos que contienen una palabra clave dada. Dese un algoritmo eficiente para hallar el conjunto de documentos deseado.

NOTAS BIBLIOGRÁFICAS

Gray et al. [1995] y Gray et al. [1997] describen el operador cubo de datos. Los algoritmos eficientes para calcular cubos de datos se describen en Agarwal et al. [1996], Harinarayan et al. [1996] y Ross y Srivastava [1997]. Las descripciones del soporte ampliado de la agregación en SQL:1999 puede hallarse en los manuales de los productos de sistemas de bases de datos como Oracle y DB2 de IBM. Las definiciones de las funciones estadísticas pueden hallarse en los libros de texto normales de estadística como Bulmer [1979] y Ross [1999].

Witten y Frank [1999] y Han y Kamber [2000] proporcionan cobertura del nivel de los libros de texto de la recopilación de datos. Mitchell [1997] es un libro de texto clásico sobre aprendizaje de las máquinas y trata con detalle las técnicas de clasificación. Fayyad et al. [1995] presentan un extenso conjunto de artículos sobre la búsqueda de información y la recopilación de datos. Kohavi y Provost [2001] presentan un conjunto de artículos sobre aplicaciones de la recopilación de datos para el comercio electrónico.

Agrawal et al. [1993] proporcionan una primera introducción a la recopilación de datos en las bases de datos. En Agrawal et al. [1992] y Shafer et al. [1996] se describen algoritmos para el cálculo de clasificadores con conjuntos de formación de gran tamaño; el algoritmo de creación del árbol de decisión descrito en este capítulo se basa en el algoritmo SPRINT de Shafer et al. [1996]. Agrawal y Srikant [1994] fue uno de los primeros textos sobre la recopilación de reglas de asociación. Los algoritmos para la recopilación de diferentes formas de las reglas de asociación se describen en Srikant y Agrawal [1996a] y en Srikant y Agrawal [1996b]. Chakrabarti et al. [1998] describen las técnicas para la recopilación de estructuras temporales sorprendentes.

Durante mucho tiempo se ha estudiado el agrupamiento en el área de la estadística, y Jain y Dubes [1988] proporcionan cobertura del agrupamiento del nivel de los libros de texto. Ng y Han [1994] describen las técnicas del agrupamiento espacial. Las técnicas de agrupamiento para conjuntos de datos de gran tamaño se describen en Zhang et al. [1996]. Breese et al. [1998] proporcionan un análisis empírico de diferentes algoritmos para el filtrado cooperativo. Las técnicas para el filtrado cooperativo de los artículos de noticias se describen en Konstan et al. [1997].

Chakrabarti [2000] proporciona una recopilación de técnicas de recopilación de hipertexto como la clasificación y el agrupamiento de hipertexto. Chakrabarti [1999] proporciona una recopilación de búsqueda de recursos web. Las técnicas para la integración de cubos de datos con la recopilación de datos se describen en Sarawagi [2000].

Poe [1995] y Mattison [1996] proporcionan cobertura de los almacenes de datos del nivel de los libros de texto. Zhuge et al. [1995] describen el mantenimiento de las vistas en entornos de almacenes de datos.

Witten et al. [1999], Grossman y Frieder [1998] y Baeza-Yates y Ribeiro-Neto [1999] proporcionan descripciones de libro de texto de la recuperación de la información. El indexado de los documentos se trata con detalle en Witten et al. [1999]. Jones y Willet [1997] es un conjunto de artículos sobre recuperación de la información. Salton [1989] es uno de los primeros libros de texto sobre sistemas de recuperación de la información. El índice TREC (trec.nist.gov) es una prueba para la medida de la efectividad de la recuperación.

Brin y Page [1998] describen la anatomía del motor de búsqueda Google, incluida la técnica PageRank, mientras que una técnica de clasificación de nodos y de autoridades denominada HITS se describe en Kleinberg [1999]. Bharat y Henzinger [1998] presentan una mejora de la técnica de clasificación HITS. Un punto que merece la pena destacar es que el PageRank de una página se calcula independientemente de cualquier consulta, y que como consecuencia las páginas con clasificación elevada que resulta que sólo contienen algunas palabras clave sin importancia figuran entre las primeras respuestas a las consultas sobre esas palabras clave sin importancia. Por el contrario, el algoritmo HITS tiene en consideración las palabras clave de la consulta al calcular el prestigio, pero tiene un coste más elevado para la respuesta de las consultas.

HERRAMIENTAS

Se dispone de gran variedad de herramientas para cada una de las aplicaciones que se han estudiado en este capítulo. La mayor parte de los fabricantes de bases de datos proporcionan herramientas OLAP como parte de sus sistemas de bases de datos, o como aplicaciones complementarias. Entre ellas están las herramientas OLAP de Microsoft Corp., Oracle Express e Informix Metacube. La herramienta OLAP Arbor Essbase procede de un fabricante de software independiente. El sitio www.databeacon.com proporciona una demostración en línea de las herramientas OLAP para balizamiento de datos para empleo en la web y en orígenes de datos de archivos de texto. Muchas empresas también ofrecen herramientas de análisis especializadas para aplicaciones específicas, como la gestión de las relaciones con los clientes.

También hay una gran variedad de herramientas de recopilación de datos de propósito general que incluyen las herramientas de recopilación del SAS Institute, IBM Intelligent Miner y SGI Mineset. Se necesita una gran experiencia para aplicar las herramientas de recopilación de datos de propósito general para aplicaciones específicas. En consecuencia, se han desarrollado gran número de herramientas de recopilación para abordar aplicaciones especializadas. El sitio web www.kdnuggets.com proporciona un amplio directorio de software de recopilación de datos, soluciones, publicaciones, etcétera.

Los principales fabricantes de bases de datos también ofrecen productos para el almacenamiento de datos asociados con sus sistemas de bases de datos. Éstos proporcionan funcionalidad de soporte para el modelado, la limpieza, la carga y la consulta de datos. El sitio web www.dwinfocenter.org proporciona información sobre productos de almacenaje de datos.

Google (www.google.com) es un popular motor de búsqueda. Yahoo (www.yahoo.com) y el Open Directory Project (dmoz.org) proporcionan jerarquías de clasificación para los sitios web.

CAPÍTULO

23 TIPOS DE DATOS AUTOMÁTICOS Y NUEVAS APLICACIONES

Durante la mayor parte de la historia de las bases de datos, los tipos de datos almacenados en ellas eran relativamente sencillos, y esto se reflejaba en el soporte bastante limitado de los tipos de datos en versiones anteriores de SQL. En los últimos años, sin embargo, ha habido una necesidad creciente de manejar tipos de datos nuevos en las bases de datos, como los datos temporales, los datos espaciales y los datos multimedia.

Otra tendencia importante de la última década ha creado sus propios problemas: el auge de la informática móvil, que comenzó con las computadoras portátiles y las agendas de bolsillo y, en tiempos más recientes, ha seguido para incluir los teléfonos móviles con computadoras incluidas y una gran variedad de computadoras *portátiles,* que se emplean cada vez más en aplicaciones comerciales.

En este capítulo se estudian varios tipos nuevos de datos y también se estudian los problemas de las bases de datos relacionados con la informática móvil.

23.1. MOTIVACIÓN

Antes de abordar a fondo cada uno de los temas se resumirá la motivación de cada uno de estos tipos de datos y algunos problemas importantes del trabajo con ellos.

- **Datos temporales.** La mayor parte de los sistemas de bases de datos modelan el estado actual del mundo, por ejemplo, los clientes actuales, los estudiantes actuales y los cursos que se están ofertando. En muchas aplicaciones es muy importante almacenar y recuperar la información sobre estados anteriores. La información histórica puede incorporarse de manera manual en el diseño del esquema. No obstante, la tarea se simplifica mucho con el soporte de los datos temporales por la base de datos, lo cual se estudia en el Apartado 23.2.
- **Datos espaciales.** Entre los datos espaciales están los **datos geográficos**, como los datos y la información asociada a ellos, y los **datos de diseño asistido por computadora**, como los diseños de circuitos integrados o los diseños de edificios. Las aplicaciones de datos espaciales en un principio almacenaban los datos como archivos de un sistema de archivos, igual que las aplicaciones de negocios de las primeras generaciones. Pero, a medida que la complejidad y el volumen de los datos, y el número de usuarios, han aumentado, se han probado insuficientes para las necesidades de muchas aplicaciones que utilizan datos espaciales los enfoques ad hoc para almacenar y recuperar los datos en un sistema de archivos.

  Las aplicaciones de datos espaciales necesitan los servicios ofrecidos por los sistemas de bases de datos, en especial, la posibilidad de almacenar y consultar de manera eficiente grandes cantidades de datos. Algunas aplicaciones también pueden necesitar otras características de las bases de datos, como las actualizaciones atómicas de parte de los datos almacenados, la durabilidad y el control de la concurrencia. En el Apartado 23.3 se estudian las ampliaciones de los sistemas de bases de datos tradicionales necesarias para que soporten los datos espaciales.
- **Datos multimedia.** En el Apartado 23.4 se estudian las características necesarias en los sistemas de bases de datos que almacenan datos multimedia como los datos de imágenes, de vídeo o de audio. La principal característica que diferencia a los datos de vídeo y de audio es que la exhibición de éstos exige la recuperación a una velocidad predeterminada constante; por tanto, esos datos se denominan **datos de medios continuos**.
- **Bases de datos móviles.** En el Apartado 23.5 se estudian los requisitos para las bases de datos de la nueva generación de sistemas informáticos portátiles, como las computadoras portátiles y los dispositivos informáticos de bolsillo, que se conectan con las estaciones base mediante redes de

comunicación digitales inalámbricas. Estas computadoras necesitan poder operar mientras se hallan desconectadas de la red, a diferencia de los sistemas distribuidos de bases de datos estudiados en el Capítulo 19. También tienen una capacidad de almacenamiento limitada y, por tanto, necesitan técnicas especiales para la administración de la memoria.

23.2. EL TIEMPO EN LAS BASES DE DATOS

Las bases de datos modelan el estado de algún aspecto del mundo real externo a ellas. Generalmente, las bases de datos sólo modelan un estado —el estado actual— del mundo real, y no almacenan información sobre estados anteriores excepto, quizás, como registro para la auditoría. Cuando se modifica el estado del mundo real, se actualiza la base de datos y se pierde la información sobre el estado anterior. Sin embargo, en muchas aplicaciones es importante almacenar y recuperar información sobre estados anteriores. Por ejemplo, una base de datos sobre pacientes debe almacenar información sobre el historial médico de cada paciente. El sistema de control de una fábrica puede almacenar información sobre las lecturas actuales y anteriores de los sensores de la fábrica para su análisis. Las bases de datos que almacenan información sobre los estados del mundo real a lo largo del tiempo se denominan **bases de datos temporales**.

Al considerar el problema del tiempo en los sistemas de bases de datos se distingue entre el tiempo medido por el sistema y el tiempo observado en el mundo real. El **momento válido** de un hecho es el conjunto de intervalos de tiempo en el que el hecho es cierto en el mundo real. El **momento de transacción** de un hecho es el intervalo de tiempo en el que el hecho es actual en el sistema de bases de datos. Este segundo tiempo se basa en el orden de secuenciación de las transacciones y el sistema lo genera de manera automática. Hay que tener en cuenta que los intervalos de tiempo válidos, que son un concepto del tiempo real, no pueden generarse de manera automática y se le deben proporcionar al sistema.

Una **relación temporal** es una relación en la que cada tupla tiene un tiempo asociado en que es verdadera; el tiempo puede ser tiempo válido o tiempo de transacción. Por supuesto, tanto el momento válido como el momento de transacción pueden almacenarse, en cuyo caso la relación se denomina **relación bitemporal**. La Figura 23.1 muestra un ejemplo de relación temporal. Para simplificar la representación cada tupla tiene asociado un único intervalo temporal; así, cada tupla se representa una vez para cada intervalo de tiempo disjunto en que es cierta. Los intervalos se muestran aquí como pares de atributos *de* y *a*; una implementación real tendría un tipo estructurado, acaso denominado *Intervalo*, que contuviera los dos campos. Obsérvese que algunas de las tuplas tienen un asterisco en la columna temporal *a*; esos asteriscos indican que la tupla es verdadera hasta que se modifique el valor de la columna temporal *a*; así, la tupla es cierta en el momento actual. Aunque los tiempos se muestran en forma textual, se almacenan internamente de una manera más compacta, como el número de segundos desde algún momento de una fecha fija (como las 12:00 am, 1 de enero, 1900) que puede volver a traducirse a la forma textual normal.

23.2.1. Especificación del tiempo en SQL

El estándar SQL define los tipos **date**, **time** y **timestamp**. El tipo **date** contiene cuatro cifras para los años (1 – 9999), dos para los meses (1 – 12) y dos más para los días (1 – 31). El tipo **time** contiene dos cifras para la hora, dos para los minutos y otras dos para los segundos, además de cifras decimales opcionales. El campo

Número-cuenta	*Nombre-oficina*	*Saldo*	*De*		*A*	
C-101	Centro	500	9:00	1/1/1999	11:30	24/1/1999
C-101	Centro	100	11:30	24/1/1999	*	
C-215	Becerril	700	15:30	2/6/2000	10:00	8/8/2000
C-215	Becerril	900	10:00	8/8/2000	8:00	5/9/2000
C-215	Becerril	700	8:00	5/9/2000	*	
C-217	Galapagar	750	11:00	5/7/1999	16:00	1/5/2000

FIGURA 23.1. La relación temporal *cuenta*.

de los segundos puede superar el valor de sesenta para tener en cuenta los segundos extra que se añaden algunos años para corregir las pequeñas variaciones de velocidad en la rotación de la Tierra. El tipo **timestamp** contiene los campos de **date** y de **time** con seis cifras decimales para el campo de los segundos.

Como las diferentes partes del mundo tienen horas locales diferentes suele darse la necesidad de especificar la zona horaria junto con la hora. La **hora universal coordinada (Universel Temps Coordoné, UTC)** es un punto estándar de referencia para la especificación de la hora y las horas locales se definen como diferencias respecto a UTC. (La abreviatura estándar es UTC, en lugar de UCT, ya que es una abreviatura de «tiempo universal coordinado (Universal Coordinated Time)» escrito en francés como *universel temps coordonné*.) SQL también soporta dos tipos, **time with time zone y timestamp with time zone**, que especifican la hora como la hora local más la diferencia de la hora local respecto de UTC. Por ejemplo, la hora podría expresarse en términos de la hora estándar del Este de EE.UU. (U.S. Eastern Standard Time), con una diferencia de –6:00, ya que la hora estándar del Este de EE.UU. va retrasada seis horas respecto de UTC.

SQL soporta un tipo denominado **interval**, que permite hacer referencia a un periodo de tiempo como puede ser «1 día» o «2 días y 5 horas», sin especificar la hora concreta en que comienza el periodo. Este concepto se diferencia del concepto de intervalo utilizado anteriormente, que hace referencia a un intervalo de tiempo con horas de comienzo y de final específicos[1].

23.2.2. Lenguajes para consultas temporales

Las relaciones de base de datos sin información temporal se denominan a veces **relaciones instantáneas**, ya que reflejan el estado del mundo real en una instantánea. Así, una instantánea de una relación temporal en un momento del tiempo t es el conjunto de tuplas de la relación que son ciertas en el momento t, con los atributos de intervalos de tiempo eliminados. La operación instantánea para una relación temporal da la instantánea de la relación en un momento especificado (o en el momento actual, si no se especifica el momento).

Una **selección temporal** es una selección que implica a los atributos de tiempo; una **proyección temporal** es una proyección en la que las tuplas de la proyección heredan los valores de tiempo de las tuplas de la relación original. Una **reunión temporal** es una reunión en que los valores temporales de las tuplas del resultado son la intersección de los valores temporales de las tuplas de las que proceden. Si los valores temporales no se intersecan, esas tuplas se eliminan del resultado.

Los predicados *precede*, *solapa* y *contiene* pueden aplicarse a los intervalos; sus significados deben quedar claros. La operación *intersección* puede aplicarse a dos intervalos, para dar un único intervalo (posiblemente vacío). Sin embargo, la unión de dos intervalos puede que no sea un solo intervalo.

Las dependencias funcionales deben utilizarse con cuidado en las relaciones temporales. Aunque puede que el número de cuenta determine funcionalmente el saldo en cualquier momento, evidentemente el saldo puede cambiar con el tiempo. Una **dependencia funcional temporal** $X \rightarrow^{T} Y$ es válida para un esquema de relación R si, para todos los casos legales r de R, todas las instantáneas de r satisfacen la dependencia funcional $X \rightarrow Y$.

Se han realizado varias propuestas para ampliar SQL para mejorar su soporte de los datos temporales. SQL:1999 Parte 7 (SQL/Temporal), que actualmente se halla en desarrollo, es el estándar propuesto para las extensiones temporales de SQL.

23.3. DATOS ESPACIALES Y GEOGRÁFICOS

El soporte de los datos espaciales en las bases de datos es importante para el almacenaje, indexado y consulta eficientes de los datos basados en las posiciones espaciales. Por ejemplo, supóngase que se desea almacenar un conjunto de polígonos en una base de datos y consultar la base de datos para hallar todos los polígonos que intersecan un polígono dado. No se pueden utilizar las estructuras estándares de índices como los árboles B o los índices asociativos para responder de manera eficiente esas consultas. El procesamiento eficiente de esas consultas necesita estructuras de índices de finalidades especiales, como los árboles R (que se estudiarán posteriormente).

Dos tipos de datos espaciales son especialmente importantes:

- Los **datos de diseño asistido por computadora (Computer-Aided-Design, CAD)**, que incluyen información espacial sobre el modo en que los objetos —como los edificios, los coches o los avio-

[1] Muchos investigadores de bases de datos temporales consideran que este tipo de datos debería haberse denominado **span**, ya que no especifica un momento inicial ni un momento final exactos, sino tan sólo el tiempo transcurrido entre los dos.

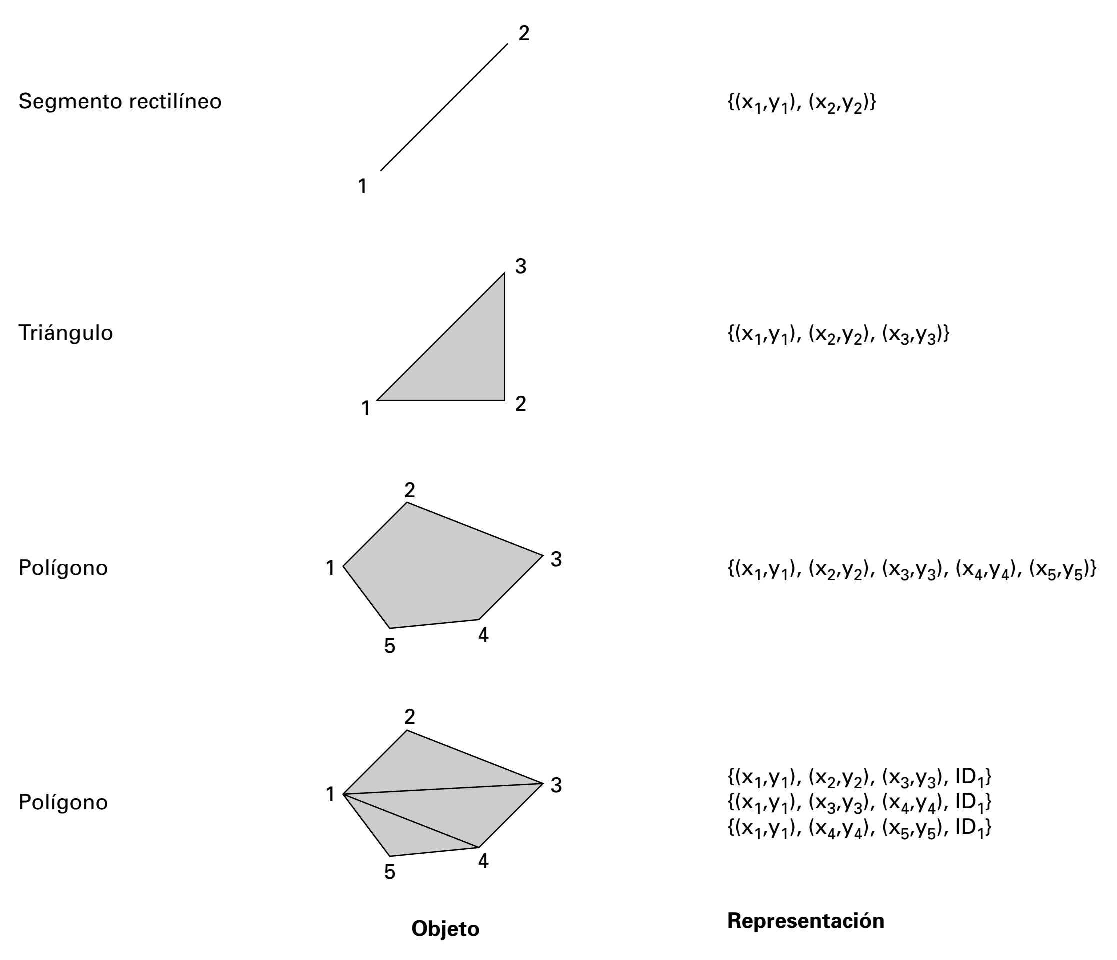


FIGURA 23.2. Representación de estructuras geométricas.

nes— están construidos. Otros ejemplos importantes de bases de datos de diseño asistido por computadora son los diseños de circuitos integrados y de dispositivos electrónicos.

- Los **datos geográficos** como los mapas de carreteras, los mapas de uso de la tierra, los mapas topográficos, los mapas políticos que muestran fronteras, los mapas catastrales, etcétera.

Los **sistemas de información geográfica** son bases de datos de propósito especial adaptadas para el almacenamiento de datos geográficos. El soporte para los datos geográficos se ha añadido a muchos sistemas de bases de datos, como IBM DB2 Spatial Extender, Informix Spatial Datablade u Oracle Spatial.

23.3.1. Representación de la información geométrica

La Figura 23.2 muestra el modo en que se pueden representar de manera normalizada en las bases de datos varias estructuras geométricas. Hay que destacar aquí que la información geométrica puede representarse de varias maneras diferentes, de las que sólo se describen algunas.

Un *segmento rectilíneo* puede representarse mediante las coordenadas de sus extremos. Por ejemplo, en una base de datos de mapas, las dos coordenadas de un punto serían su latitud y su longitud. Una *línea poligonal* (también denominada *línea quebrada*) consiste en una secuencia conectada de segmentos rectilíneos, y pueden representarse mediante una lista que contenga las coordenadas de los extremos de los segmentos, en secuencia. Se puede representar aproximadamente una curva arbitraria mediante líneas poligonales, dividiendo la curva en una serie de segmentos. Esta representación resulta útil para elementos bidimensionales como las carreteras; en este caso, la anchura de la carretera es lo bastante pequeña en relación con el tamaño de todo el mapa que puede considerarse bidimensional. Algunos sistemas también soportan como primitivas los *arcos de circunferencia*, lo que permite que las curvas se representen como secuencias de arcos.

Los *polígonos* pueden representarse indicando sus vértices en orden, como en la Figura 23.2 [2]. La lista de

[2] Algunas referencias utilizan el término polígono cerrado para hacer referencia a lo que aquí se denominan polígonos y se refieren a las líneas poligonales abiertas como polígonos abiertos.

los vértices especifica la frontera de cada región poligonal. En una representación alternativa cada polígono puede dividirse en un conjunto de triángulos, como se muestra en la Figura 23.2. Este proceso se denomina **triangulación**, y se puede triangular cualquier polígono. Se concede un identificador a cada polígono complejo, y cada uno de los triángulos en los que se divide lleva el identificador del polígono. Los círculos y las elipses pueden representarse por los tipos correspondientes, o aproximarse mediante polígonos.

Las representaciones de las líneas poligonales o de los polígonos basadas en listas suelen resultar convenientes para el procesamiento de consultas. Esas representaciones que no están en la primera forma normal se utilizan cuando están soportadas por la base de datos subyacente. Para que se puedan utilizar tuplas de tamaño fijo (en la primera forma normal) para la representación de líneas poligonales se puede dar a la línea poligonal o a la curva un identificador, y se puede representar cada segmento como una tupla separada que también lleva el identificador de la línea poligonal o de la curva. De manera parecida, la representación triangulada de los polígonos permite una representación relacional de los polígonos en su primera forma normal.

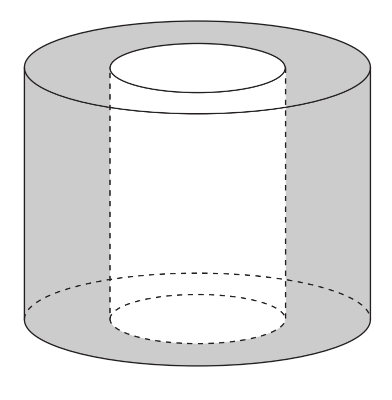
La representación de los puntos y de los segmentos rectilíneos en el espacio tridimensional es parecida a su representación en el espacio bidimensional, siendo la única diferencia que los puntos tienen un componente z adicional. De manera parecida, la representación de las figuras planas —como los triángulos, los rectángulos y otros polígonos— no cambia mucho cuando se pasa a tres dimensiones. Los tetraedros y los paralelepípedos pueden representarse de la misma manera que los triángulos y los rectángulos. Se pueden representar poliedros arbitrarios dividiéndolos en tetraedros, igual que se triangulan los polígonos. También se pueden representar indicando las caras, cada una de las cuales es, en sí misma, un polígono, junto con una indicación del lado de la cara que está por dentro del poliedro.

23.3.2. Bases de datos de diseño

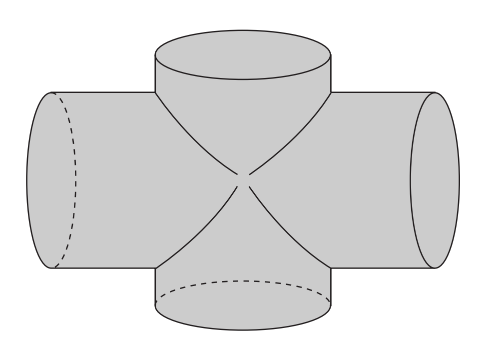
Los **sistemas de diseño asistido por computadora (Computer-Aided-Design, CAD)** tradicionalmente almacenaban los datos en la memoria durante su edición u otro tipo de procesamiento y los volvían a escribir en archivos al final de la sesión de edición. Entre los inconvenientes de este esquema están el coste (la complejidad de programación y el coste temporal) de transformar los datos de una forma a otra, y la necesidad de leer todo un archivo aunque sólo sea necesaria una parte. Para los diseños de gran tamaño, como el diseño de circuitos integrados a gran escala o el diseño de todo un avión, puede que resulte imposible guardar en la memoria el diseño completo. Los diseñadores de bases de datos orientadas a objetos estaban motivados en gran parte por las necesidades de los sistemas de CAD. Las bases de datos orientadas a objetos representan los componentes del diseño como objetos y las conexiones entre los objetos indican el modo en que está estructurado el diseño.

Los objetos almacenados en las bases de datos de diseño suelen ser objetos geométricos. Entre los objetos geométricos bidimensionales sencillos están los puntos, las líneas, los triángulos, los rectángulos y los polígonos en general. Los objetos bidimensionales complejos pueden formarse a partir de objetos sencillos mediante las operaciones de unión, intersección y diferencia. De manera parecida, los objetos tridimensionales complejos pueden formarse a partir de objetos más sencillos como las esferas, los cilindros y los paralelepípedos mediante las operaciones unión, intersección y diferencia, como en la Figura 23.3. Las superficies tridimensionales también pueden representarse mediante **modelos de alambres**, que esencialmente modelan las superficies como conjuntos de objetos más sencillos, como segmentos rectilíneos, triángulos y rectángulos.

Las bases de datos de diseño también almacenan información no espacial sobre los objetos, como el material del que están construidos. Se suele poder modelar



(a) Diferencia de cilindros



(b) Unión de cilindros

FIGURA 23.3. Objetos tridimensionales complejos.

esa información mediante técnicas estándar de modelado de datos. Aquí se centrará la atención únicamente en los aspectos espaciales.

Al diseñar hay que realizar varias operaciones espaciales. Por ejemplo, puede que el diseñador desee recuperar la parte del diseño que corresponde a una región de interés determinada. Las estructuras espaciales de índices, estudiadas en el Apartado 23.3.5, resultan útiles para estas tareas. Las estructuras espaciales de índices son multidimensionales, trabajan con datos de dos y de tres dimensiones, en vez de trabajar solamente con la sencilla ordenación unidimensional que proporcionan los árboles B^+.

Las restricciones de integridad espacial, como «dos tuberías no deben estar en la misma ubicación», son importantes en las bases de datos de diseño para evitar errores por interferencias. Estos errores suelen producirse si el diseño se realiza a mano y sólo se detectan al construir un prototipo. En consecuencia, estos errores pueden resultar costosos de reparar. El soporte de las bases de datos para las restricciones de integridad espacial ayuda a los usuarios a evitar los errores de diseño, con lo que hacen que el diseño sea consistente. La implementación de esas verificaciones de integridad depende una vez más de la disponibilidad de estructuras multidimensionales de índices eficientes.

23.3.3. Datos geográficos

Los datos geográficos son de naturaleza espacial, pero se diferencian de los datos de diseño en ciertos aspectos. Los mapas y las imágenes de satélite son ejemplos típicos de datos geográficos. Los mapas pueden proporcionar no sólo información sobre la ubicación —sobre fronteras, ríos y carreteras, por ejemplo— sino también información mucho más detallada asociada con la ubicación, como la elevación, el tipo de suelo, el uso de la tierra y la cantidad anual de lluvia.

Los **datos geográficos** pueden clasificarse en dos tipos:

- Los **datos por líneas**. Estos datos consisten en mapas de bits o en mapas de píxeles en dos o más dimensiones. Un ejemplo típico de imagen de líneas bidimensional son las imágenes de satélite de cobertura nubosa, en las que cada píxel almacena la visibilidad de nubes en un área concreta. Estos datos pueden ser tridimensionales, por ejemplo, la temperatura a diferentes altitudes en distintas regiones, también medidas con la ayuda de un satélite. El tiempo puede formar otra dimensión, por ejemplo, las medidas de la temperatura superficial en diferentes momentos. Las bases de datos de diseño no suelen almacenar datos por líneas.
- Los **datos vectoriales**. Los datos vectoriales están formados a partir de objetos geométricos básicos como los puntos, los segmentos rectilíneos, los triángulos y otros polígonos en dos dimensiones y los cilindros, las esferas, los paralelepípedos y otros poliedros en tres dimensiones.

Los datos de los mapas suelen representarse en formato vectorial. Los ríos y las carreteras pueden representarse como uniones de varios segmentos rectilíneos. Los estados y los países pueden representarse como polígonos. La información topológica como la altura puede representarse mediante una superficie dividida en polígonos que cubren las regiones de igual altura, con un valor de altura asociado a cada polígono.

23.3.3.1. Representación de los datos geográficos

Los accidentes geográficos, como los estados y los grandes lagos, se representan como polígonos complejos. Algunos accidentes, como los ríos, pueden representarse como curvas complejas o como polígonos complejos, en función de si su anchura es importante o no.

La información geográfica relativa a las regiones, como la cantidad anual de lluvia, puede representarse como un array, es decir, en forma de líneas. En aras de la eficiencia del espacio, el array puede almacenarse de manera comprimida. En el Apartado 23.3.5 se estudia una representación alternativa de estas arrays mediante una estructura de datos denominada *árbol cuadrático*.

Como se indicó en el Apartado 23.3.3, se puede representar la información regional en forma vectorial empleando polígonos, cada uno de los cuales es una región en la que el valor del array es el mismo. La representación vectorial es más compacta que la de líneas en algunas aplicaciones. También es más precisa para algunas tareas, como el dibujo de carreteras, en que la división de la región en píxeles (que pueden ser bastante grandes) lleva a una pérdida de precisión en la información de ubicación. No obstante, la representación vectorial resulta inadecuada para las aplicaciones en que los datos se basan en líneas de manera intrínseca, como las imágenes de satélite.

23.3.3.2. Aplicaciones de los datos geográficos

Las bases de datos geográficas tienen gran variedad de usos, incluidos los servicios de mapas en línea, los sistemas de navegación para vehículos, la información de redes de distribución para las empresas de servicios públicos como son los sistemas de telefonía, electricidad y suministro de agua y la información de uso de la tierra para ecologistas y planificadores.

Los servicios de mapas de carreteras basados en la web constituyen una aplicación muy utilizada de datos de mapas. En su nivel más sencillo estos sistemas pueden utilizarse para generar mapas de carreteras en línea de la región deseada. Una ventaja importante de los mapas interactivos es que resulta sencillo dimensionar los mapas al tamaño deseado, es decir, acercarse y alejarse para ubicar los accidentes importantes. Los servicios de mapas de carretera también almacenan infor-

mación sobre carreteras y servicios, como el trazado de las carreteras, los límites de velocidad, las condiciones de las vías, las conexiones entre carreteras y los tramos de sentido único. Con esta información adicional sobre las carreteras se pueden utilizar los mapas para obtener indicaciones para desplazarse de un sitio a otro y para localizar, por ejemplo, hoteles, gasolineras o restaurantes con las ofertas y gamas de precios deseadas.

Los sistemas de navegación para los vehículos son sistemas montados en automóviles que proporcionan mapas de carreteras y servicios para la planificación de los viajes. Un añadido útil a los sistemas de información geográfica móviles como los sistemas de navegación de los vehículos es una unidad del **sistema de posicionamiento global (Global Positioning System, GPS**), que utiliza la información emitida por los satélites GPS para hallar la ubicación actual con una precisión de decenas de metros. Con estos sistemas el conductor no puede perderse nunca[3]: la unidad GPS halla la ubicación en términos de latitud, longitud y elevación y el sistema de navegación puede consultar la base de datos geográfica para hallar el lugar en que se encuentra y la carretera en que se halla el vehículo.

Las bases de datos geográficas para información de utilidad pública se están volviendo cada vez más importantes a medida que crece la red de cables y tuberías enterrados. Sin mapas detallados las obras realizadas por una empresa de servicio público pueden dañar los cables de otra, lo que daría lugar a una interrupción del servicio a gran escala. Las bases de datos geográficas, junto con los sistemas precisos de determinación de la posición, pueden ayudar a evitar estos problemas.

Hasta ahora se ha explicado el motivo de que las bases de datos espaciales resulten útiles. En el resto del apartado se estudiarán los detalles técnicos, como la representación y el indexado de la información espacial.

23.3.4. Consultas espaciales

Hay varios tipos de consultas que implican a las ubicaciones espaciales.

- Las **consultas de proximidad** solicitan objetos que se hallen cerca de una ubicación especificada. La consulta para hallar todos los restaurantes que se hallan a menos de una distancia dada de un determinado punto es un ejemplo de consulta de proximidad. La **consulta de vecino más próximo** solicita el objeto que se halla más próximo al punto especificado. Por ejemplo, puede que se desee hallar la gasolinera más cercana. Obsérvese que esta consulta no tiene que especificar un límite para la distancia y, por tanto, se puede formular aunque no se tenga idea de la distancia a la que se halla la gasolinera más próxima.
- Las **consultas regionales** tratan de regiones espaciales. Estas consultas pueden preguntar por objetos que se hallen parcial o totalmente en el interior de la región especificada. Un ejemplo es la consulta para hallar todas las tiendas minoristas dentro de los límites geográficos de una ciudad dada.
- Puede que las consultas también soliciten **intersecciones** y **uniones** de regiones. Por ejemplo, dada la información regional, como pueden ser la lluvia anual y la densidad de población, una consulta puede solicitar todas las regiones con una baja cantidad de lluvia anual y una elevada densidad de población.

Las consultas que calculan las intersecciones de regiones pueden considerarse como si calcularan la **reunión espacial** de dos relaciones espaciales —por ejemplo, una que represente la cantidad de lluvia y otra que represente la densidad de población— con la ubicación en el papel de atributo de reunión. En general, dadas dos relaciones, cada una de las cuales contiene objetos espaciales, la reunión espacial de las dos relaciones genera pares de objetos que se intersectan o las regiones de intersección de esos pares.

Varios algoritmos de reunión calculan de manera eficiente las reuniones espaciales de datos vectoriales. Aunque se pueden utilizar las reuniones de bucles anidados o de bucles anidados indexados (con índices espaciales), las reuniones de asociación y las reuniones por mezcla-ordenación no pueden utilizarse con datos espaciales. Los investigadores han propuesto técnicas de reunión basadas en el recorrido coordinado de las estructuras espaciales de los índices de las dos relaciones. Véanse las notas bibliográficas para obtener más información.

En general, las consultas de datos espaciales pueden tener una combinación de requisitos espaciales y no espaciales. Por ejemplo, puede que se desee averiguar el restaurante más cercano que tenga menú vegetariano y que cueste menos de diez euros por comida.

Dado que los datos espaciales son inherentemente gráficos, se suelen consultar mediante un lenguaje gráfico de consulta. El resultado de esas consultas también se muestra gráficamente, en vez de mostrarse en tablas. El usuario puede realizar varias operaciones con la interfaz, como escoger el área que desea ver (por ejemplo, apuntando y pulsando en los barrios del oeste de Arganzuela), acercarse y alejarse, escoger lo que desea mostrar de acuerdo con las condiciones de selección (por ejemplo, casas con más de tres habitaciones), superponer varios mapas (por ejemplo, las casas con más de tres habitaciones superpuestas sobre un mapa que muestre las zonas con bajas tasas de delincuencia), etcétera. La interfaz gráfica constituye la parte visible para el usuario. Se han propuesto extensiones de SQL para permitir que las bases de datos relacionales almacenen y recu-

[3] Bueno, ¡casi nunca!

peren información espacial de manera eficiente y también permitir que las consultas mezclen las condiciones espaciales con las no espaciales. Las extensiones incluyen la autorización de tipos de datos abstractos como las líneas, los polígonos y los mapas de bits y la autorización de condiciones espaciales como *contiene* o *solapa*.

23.3.5. Indexado de los datos espaciales

Los índices son necesarios para el acceso eficiente a los datos espaciales. Las estructuras de índices tradicionales, como los índices de asociación y los árboles B, no resultan adecuadas, ya que sólo trabajan con datos unidimensionales, mientras que los datos espaciales suelen ser de dos o más dimensiones.

23.3.5.1. Los árboles k-d

Para comprender el modo de indexar los datos espaciales que constan de dos o más dimensiones se considera en primer lugar el indexado de los puntos de los datos unidimensionales. Las estructuras arbóreas, como los árboles binarios y los árboles B, operan dividiendo el espacio en partes más pequeñas de manera sucesiva. Por ejemplo, cada nodo interno de un árbol binario divide un intervalo unidimensional en dos. Los puntos que quedan en la partición izquierda van al subárbol izquierdo; los puntos que quedan en la partición de la derecha van al subárbol derecho. En los árboles binarios equilibrados la partición se escoge de modo que, aproximadamente, la mitad de los puntos almacenados en el subárbol caigan en cada partición. De manera parecida, cada nivel de un árbol B divide un intervalo unidimensional en varias partes.

Se puede utilizar esa intuición para crear estructuras arbóreas para el espacio bidimensional, así como para espacios de más dimensiones. Una estructura arbórea denominada **árbol k-d** fue una de las primeras estructuras utilizadas para la indexación en varias dimensiones. Cada nivel de un árbol k-d divide el espacio en dos. La división se realiza según una dimensión en el nodo del nivel superior del árbol, según otra dimensión en los nodos del nivel siguiente, etcétera, alternando cíclicamente las dimensiones. La división se realiza de tal modo que, en cada nodo, aproximadamente la mitad de los puntos almacenados en el subárbol cae a un lado y la otra mitad al otro. La división se detiene cuando un nodo tiene menos puntos que un valor máximo dado. La Figura 23.4 muestra un conjunto de puntos en el espacio bidimensional y una representación en árbol k-d de ese conjunto de puntos. Cada línea corresponde a un nodo del árbol, y el número máximo de puntos en cada nodo hoja se ha definido como uno. Cada línea de la figura (aparte del marco exterior) corresponde a un nodo del árbol k-d. La numeración de las líneas en la figura indica el nivel del árbol en el que aparece el nodo correspondiente.

El **árbol k-d-B** extiende el árbol k-d para permitir varios nodos hijo por cada nodo interno, igual que los árboles B extienden los árboles binarios, para reducir la altura del árbol, los árboles k-d-B están mejor adaptados al almacenamiento secundario que los árboles k-d.

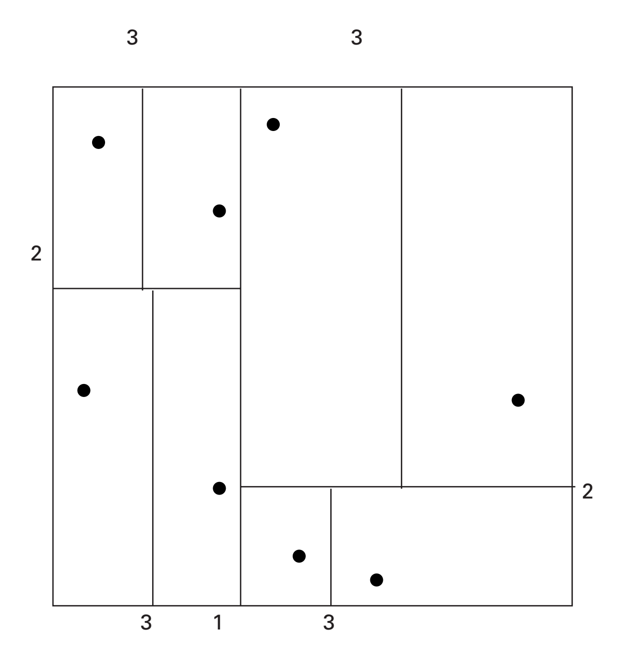


FIGURA 23.4. División del espacio por un árbol k-d.

23.3.5.2. Árboles cuadráticos

Una representación alternativa de los datos bidimensionales son los **árboles cuadráticos**. En la Figura 23.5 aparece un ejemplo de la división del espacio mediante un árbol cuadrático. El conjunto de puntos es el mismo que en la Figura 23.4. Cada nodo de un árbol cua-

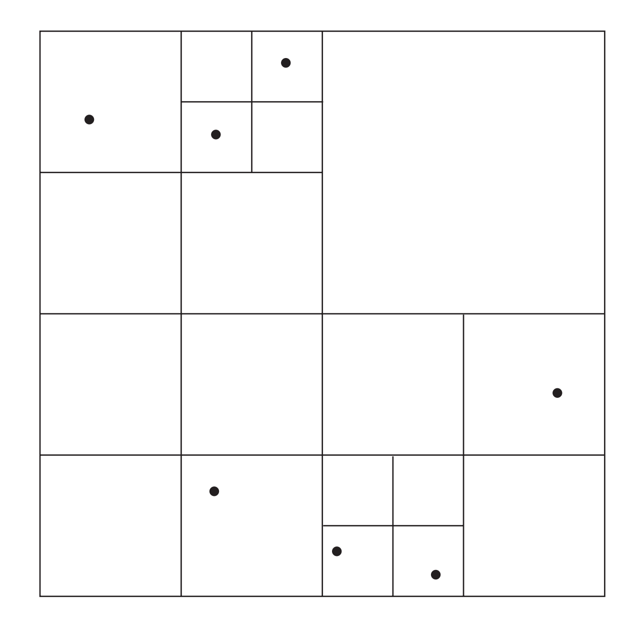


FIGURA 23.5. División del espacio por un árbol cuadrático.

drático está asociado con una región rectangular del espacio. El nodo superior está asociado con todo el espacio objetivo. Cada nodo que no sea un nodo hoja del árbol cuadrático divide su región en cuatro cuadrantes del mismo tamaño y, a su vez, cada uno de esos nodos tiene cuatro nodos hijo correspondientes a los cuatro cuadrantes. Los nodos hoja tienen un número de puntos que varía entre cero y un número máximo fijado. A su vez, si la región correspondiente a un nodo tiene más puntos que el máximo fijado, se crean nodos hijo para ese nodo. En el ejemplo de la Figura 23.5, el número máximo de puntos de cada nodo hoja está fijado en uno.

Este tipo de árbol cuadrático se denomina **árbol cuadrático PR**, para indicar que almacena los puntos y que la división del espacio se basa en regiones, en vez de en el conjunto real de puntos almacenados. Se pueden utilizar **árboles cuadráticos regionales** para almacenar información de arrays (de líneas). Cada nodo de los árboles cuadráticos regionales es un nodo hoja si todos los valores del array de la región que abarca son iguales. En caso contrario, se vuelve a subdividir en cuatro nodos hijo con la misma área y es, por tanto, un nodo interno. Cada nodo del árbol cuadrático regional corresponde a un subarray de valores. Los subarrays correspondientes a las hojas contienen un solo elemento del array o varios, todos ellos con el mismo valor.

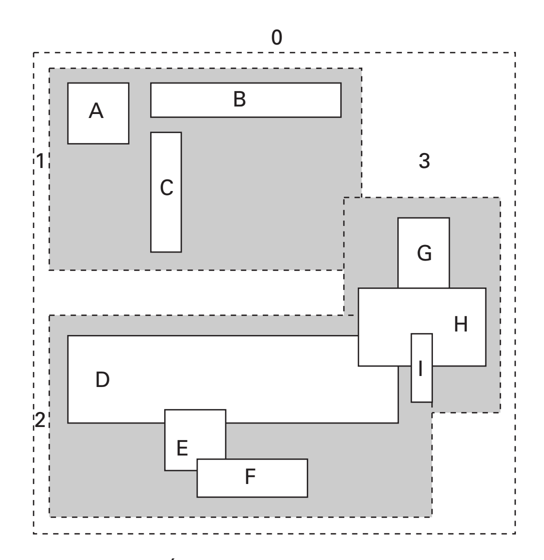
El indexado de los segmentos rectilíneos y de los polígonos presenta problemas nuevos. Hay extensiones de los árboles k-d y de los árboles cuadráticos para esta labor. No obstante, un segmento rectilíneo o un polígono puede cruzar una línea divisoria. Si lo hace, hay que dividirlo y representarlo en cada uno de los subárboles en que aparezcan sus fragmentos. La aparición múltiple de un segmento lineal o de un polígono puede dar lugar a ineficiencias en el almacenamiento, así como a ineficiencias en las consultas.

23.3.5.3. Árboles R

Una estructura de almacenamiento denominada **árbol R** resulta útil para el indexado de los rectángulos y de otros polígonos. Un árbol R es una estructura arbórea equilibrada con los polígonos indexados almacenados en los nodos hoja, de manera parecida a los árboles B^+. No obstante, en lugar de un rango de valores, se asocia una **caja límite** con cada nodo del árbol. La caja límite de un nodo hoja es el rectángulo mínimo paralelo a los ejes que contiene todos los objetos almacenados en el nodo hoja. La caja límite de los nodos internos, de manera parecida, es el rectángulo mínimo paralelo a los ejes que contiene las cajas límite de sus nodos hijo. La caja límite de un polígono viene definida, de manera parecida, como el rectángulo mínimo paralelo a los ejes que contiene al polígono.

Cada nodo interno almacena las cajas límite de los nodos hijo junto con los punteros para los nodos hijo. Cada nodo hijo almacena los polígonos indexados y puede que, opcionalmente, almacene las cajas límite de esos polígonos; las cajas límite ayudan a acelerar las comprobaciones de solapamientos de los rectángulos con los polígonos indexados —si el rectángulo de una consulta no se solapa con la caja límite de un polígono, o no se puede solapar tampoco el polígono. (Si los polígonos indexados son rectángulos, por supuesto no hace falta almacenar las cajas límite, ya que son idénticas a los rectángulos.)

La Figura 23.6 muestra el ejemplo de un conjunto de rectángulos (dibujados con línea continua) y de sus cajas límite (dibujadas con línea discontinua) de los nodos de un árbol R para el conjunto de rectángulos. Hay que tener en cuenta que las cajas límite se muestran con espacio adicional en su interior, para hacerlas visibles en el dibujo. En realidad, las cajas son menores y se ajustan fielmente a los objetos que contienen; es decir, cada cara de una caja límite *C* toca como mínimo uno de los objetos o de las cajas límite que se contienen en *C*.



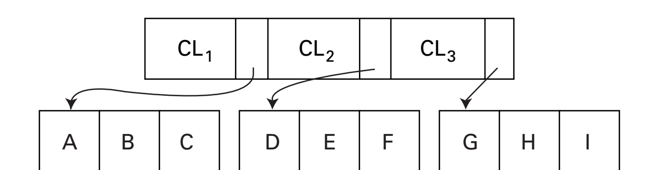


FIGURA 23.6. Árbol R.

El árbol R en sí se halla a la derecha de la Figura 23.6. La figura hace referencia a las coordenadas de la caja límite *i* como CL_i.

Ahora se verá el modo de implementar las operaciones de búsqueda, inserción y eliminación en los árboles R.

- **Búsqueda:** Como muestra la figura, las cajas límite con nodos hermanos pueden solaparse; en los árboles B$^+$, los árboles k-d y los árboles cuadráticos, sin embargo, los rangos no se solapan. La búsqueda de polígonos que contengan un punto, por tanto, tiene que seguir *todos* los nodos hijo cuyas cajas limites asociadas contengan el punto; en consecuencia, puede que haya que buscar por varios caminos. De manera parecida, una consulta para buscar todos los polígonos que intersecten con uno dado tiene que bajar por todos los nodos en que el rectángulo asociado intersecte al polígono.
- **Inserción:** Cuando se inserta un polígono en un árbol R se selecciona un nodo hoja para que lo guarde. Lo ideal sería seleccionar un nodo hoja que tuviera espacio para guardar una nueva entrada, y cuya caja límite contuviera a la caja límite del polígono. Sin embargo, puede que ese nodo no exista; aunque exista, hallarlo puede resultar muy costoso, ya que no es posible hallarlo mediante un solo recorrido desde la raíz. En cada nodo interno se pueden encontrar varios nodos hijo cuyas cajas límite contengan la caja límite del polígono, y hay que explorar cada uno de esos nodos hijo. Por tanto, como norma heurística, en un recorrido desde la raíz, si alguno de los nodos hijo tiene una caja límite que contenga a la caja límite del polígono, el algoritmo del árbol R escoge una de ellas de manera arbitraria. Si ninguno de los nodos hijo satisface esta condición, el algoritmo escoge un nodo hijo cuya caja límite tenga el solapamiento máximo con la caja límite del polígono para continuar el recorrido.

  Una vez que se ha llegado al nodo hoja, si el nodo ya está lleno, el algoritmo lleva a cabo una división del nodo (y propaga hacia arriba la división si es necesario) de manera muy parecida a la inserción de los árboles B$^+$. Igual que con la inserción en los árboles B$^+$, el algoritmo de inserción de los árboles R asegura que el árbol siga equilibrado. Además, asegura que las cajas límite de los nodos hoja, así como los nodos internos, sigan siendo consistentes; es decir, las cajas límite de los nodos hoja contienen todas las cajas límites de los polígonos almacenados en el nodo hoja, mientras que las cajas límite de los nodos internos contienen todas las cajas límite de los nodos hijo.

  La principal diferencia del procedimiento de inserción con la inserción en los árboles B$^+$ radica en el modo en que se dividen los nodos. En los árboles B$^+$ es posible hallar un valor tal que la mitad de los de las entradas sea menor que el punto medio y la mitad sea mayor que el valor. Esta propiedad no se generaliza más allá de una dimensión; es decir, para más de una dimensión, no siempre es posible dividir las entradas en dos conjuntos tales que sus cajas límite no se solapen. En lugar de eso, como norma heurística, el conjunto de entradas S puede dividirse en dos conjuntos disjuntos S_1 y S_2 tales que las cajas límite de S_1 y S_2 tengan un área total mínima; otra norma heurística sería dividir las entradas en dos conjuntos S_1 y S_2 de modo que S_1 y S_2 tengan un solapamiento mínimo. Los dos nodos resultantes de la división contendrían las entradas de S_1 y S_2, respectivamente. El coste de hallar las divisiones con área total o solapamiento mínimos puede ser elevado, por lo que se utilizan normas heurísticas más económicas, como la heurística de la *división cuadrática*. (La heurística recibe el nombre del hecho de que toma el cuadrado del tiempo en el número de entradas.)

  La heurística de la **división cuadrática** funciona de esta manera: En primer lugar, selecciona un par de entradas a y b de S tales que al ponerlas en el mismo nodo den lugar a una caja límite con el máximo de espacio desaprovechado; es decir, el área de la caja límite mínima de a y b menos la suma de las áreas de a y b es máxima. La heurística sitúa las entradas a y b en los conjuntos S_1 y S_2, respectivamente.

  Luego añade de manera iterativamente las entradas restantes, una por cada iteración, a uno de los dos conjuntos S_1 o S_2. En cada iteración, para cada entrada restante e, $i_{e,1}$ denota el incremento en el tamaño de la caja límite de S_1 si e se añade a S_1 e $i_{e,2}$ denota el incremento correspondiente de S_2. En cada iteración la heurística escoge una de las entradas con la máxima diferencia entre $i_{e,1}$ e $i_{e,2}$ y la añade a S_1 si $i_{e,1}$ es menor que $i_{e,2}$, y a S_2 en caso contrario. Es decir, se escoge en cada iteración una entrada con «preferencia máxima» por S_1 o S_2. Las iteraciones se detienen cuando se han asignado todas las entradas o cuando uno de los conjuntos S_1 o S_2 tiene entradas suficientes como para que todas las demás entradas haya que añadirlas al otro conjunto de modo que los nodos creados a partir de S_1 y S_2 tengan la ocupación mínima exigida. La heurística añade luego todas las entradas no asignadas al conjunto con menos entradas.

- **Eliminación:** La eliminación puede llevarse a cabo como si fuera una eliminación de árbol B$^+$, tomando prestadas las entradas de los nodos hermanos, o mezclando nodos hermanos si un nodo se queda menos lleno de lo exigido. Un enfoque alternativo redistribuye todas las entradas de los nodos menos llenos de lo necesario a los nodos hermanos, con el objetivo de mejorar el agrupamiento de las entradas en el árbol R.

Véanse las referencias bibliográficas para obtener más detalles de las operaciones de inserción y de eliminación en los árboles R, así como de las variantes de los árboles R, denominados árboles R* o árboles R^+.

La eficiencia del almacenamiento de los árboles R es mayor que la de los árboles k-d y que la de los árboles cuadráticos, ya que cada polígono sólo se almacena una vez, y se puede asegurar que cada nodo está, como mínimo, medio lleno. No obstante, las consultas pueden resultar más lentas, ya que hay que buscar por varios caminos. Las reuniones espaciales son más sencillas con los árboles cuadráticos que con los árboles R, ya que todos los árboles cuadráticos de cada región están divididos de la misma manera. Sin embargo, debido a su mayor eficiencia de almacenamiento, y a su parecido con los árboles B, los árboles R y sus variantes se han hecho populares en los sistemas de bases de datos que soportan datos espaciales.

23.4. BASES DE DATOS MULTIMEDIA

Los datos multimedia, como las imágenes, el sonido y el vídeo —una modalidad de datos cada vez más popular— se almacenan hoy en día casi siempre fuera de las bases de datos, en sistemas de archivos. Este tipo de almacenamiento no supone ningún problema cuando el número de objetos multimedia es relativamente pequeño, ya que las características proporcionadas por las bases de datos no suelen ser importantes.

Sin embargo, las características de las bases de datos se vuelven importantes cuando el número de objetos multimedia almacenados es grande. Aspectos como las actualizaciones transaccionales, las facilidades de consulta y el indexado se vuelven importantes. Los objetos multimedia suelen tener atributos descriptivos, como los que indican su fecha de creación, su creador y la categoría a la que pertenecen. Un enfoque de la creación de una base de datos para esos objetos multimedia es utilizar las bases de datos para almacenar los atributos descriptivos y realizar un seguimiento de los archivos en los que se almacenan los objetos multimedia.

Sin embargo, el almacenamiento de los objetos multimedia fuera de la base de datos hace más difícil proporcionar la funcionalidad de la base de datos, como el indexado con base en el contenido real de datos multimedia. También puede llevar a inconsistencias, como que un archivo esté registrado en la base de datos pero que sus contenidos falten, o viceversa. Por tanto, resulta deseable almacenar los propios datos en la base de datos.

Hay que abordar varios aspectos si se pretende almacenar los datos multimedia en una base de datos.

- La base de datos debe soportar objetos de gran tamaño, ya que los datos multimedia como los vídeos pueden ocupar varios gigabytes de espacio de almacenamiento. Los objetos de mayor tamaño pueden dividirse en fragmentos menores y almacenarse en la base de datos. De manera alternativa, los objetos multimedia pueden almacenarse en un sistema de archivos, pero la base de datos puede contener un puntero hacia el objeto; el puntero suele ser un nombre de archivo. El estándar SQL/MED (MED significa Management of External Data, gestión de datos externos), que está en desarrollo, permite que los datos externos, como los archivos, se traten como si formaran parte de la base de datos. Con SQL/MED parece que los objetos son parte de la base de datos, pero pueden almacenarse externamente.

  Los formatos de los datos multimedia se estudian en el Apartado 23.4.1.
- La recuperación de algunos tipos de datos, como los de sonido y los de vídeo, tiene la exigencia de que la entrega de los datos debe realizarse a una velocidad constante garantizada. Estos datos se denominan a veces **datos isócronos** o **datos de medios continuos**. Por ejemplo, si los datos de sonido no se proporcionan a tiempo, habrá saltos en el sonido. Si los datos se proporcionan demasiado deprisa, se pueden desbordar los búferes, lo que dará lugar a la pérdida de datos. Los datos de medios continuos se estudian en el Apartado 23.4.2.
- La recuperación basada en la semejanza es necesaria en muchas aplicaciones de bases de datos multimedia. Por ejemplo, en una base de datos que almacene imágenes de huellas digitales, se proporciona una consulta de la imagen de una huella dactilar y hay que recuperar las huellas dactilares de la base de datos que sean parecidas a la huella dactilar de la consulta. Las estructuras de índices como los árboles B^+ y los árboles R no se pueden utilizar para esta finalidad; hay que crear estructuras especiales de índices. La recuperación basada en el parecido se estudia en el Apartado 23.4.3

23.4.1. Formatos de datos multimedia

Debido al gran número de bytes necesarios para representar los datos multimedia es fundamental que se almacenen y transmitan de manera comprimida. Para los datos de imágenes el formato más utilizado es *JPEG*, que recibe su nombre del organismo de normalización que lo creó, el *Joint Picture Experts Group*, el grupo conjunto de expertos en imágenes. Se pueden almacenar los datos de vídeo codificando cada fotograma de vídeo en formato JPEG, pero esa codificación supone un desperdicio, ya que los fotogramas de vídeo sucesivos suelen ser casi iguales. El *Moving Picture Experts*

Group, grupo de expertos en películas, desarrolló la serie de estándares *MPEG* para codificar los datos de vídeo y de sonido; estas codificaciones aprovechan las similitudes de las secuencias de fotogramas para conseguir un grado de compresión mayor. El estándar *MPEG-1* almacena un minuto de vídeo y sonido a treinta fotogramas por segundo en unos 12.5 megabytes (en comparación con los aproximadamente 75 megabytes sólo para vídeo en JPEG). No obstante, la codificación MPEG-1 introduce alguna pérdida de calidad del vídeo, a un nivel aproximadamente equivalente al de las cintas de vídeo VHS.

El estándar *MPEG-2* se diseñó para los sistemas de radiodifusión digitales y para los discos de vídeo digitales (DVD); sólo introduce una pérdida de calidad de vídeo despreciable. MPEG-2 comprime un minuto de vídeo y de sonido en aproximadamente 17 megabytes. Se utilizan varios estándares competidores para la codificación de sonido, entre ellos *MP3*, que significa MPEG-1 Capa 3, RealAudio y otros formatos.

23.4.2. Datos de medios continuos

Los tipos más importantes de datos de medios continuos son los datos de vídeo y los de sonido (por ejemplo, una base de datos de películas). Los sistemas de medios continuos se caracterizan por sus requisitos de entrega de información en tiempo real:

- Los datos deben entregarse lo bastante rápido como para que no haya saltos en el resultado de sonido o de vídeo.
- Los datos deben entregarse a una velocidad que no cause un desbordamiento de los búferes del sistema.
- La sincronización entre los distintos flujos de datos debe conservarse. Esta necesidad surge, por ejemplo, cuando el vídeo de una persona que habla muestra sus labios moviéndose de manera sincronizada con el sonido de su voz.

Para proporcionar los datos de manera predecible en el momento correcto a un gran número de consumidores de los datos la captura de los datos desde el disco debe coordinarse cuidadosamente. Generalmente los datos se capturan en ciclos periódicos. En cada ciclo, digamos de *n* segundos, se capturan *n* segundos de datos para cada consumidor y se almacenan en los búferes de memoria, mientras los datos capturados en el ciclo anterior se envían a los consumidores desde los búferes de memoria. El periodo de los ciclos es un compromiso: un periodo corto utiliza menos memoria pero necesita más movimientos del brazo del disco, lo que supone un desperdicio de recursos, mientras que un periodo largo reduce el movimiento del brazo del disco pero aumenta las necesidades de memoria y puede retrasar la entrega inicial de datos. Cuando llega una nueva solicitud entra en acción el **control de admisión**: es decir, el sistema comprueba si se puede satisfacer la solicitud con los recursos disponibles (en cada periodo); si es así, se admite; en caso contrario, se rechaza.

La extensa investigación realizada sobre la entrega de datos de medios continuos ha tratado aspectos como el manejo de arrays de discos y el tratamiento de los fallos de los discos. Véanse las referencias bibliográficas para obtener más detalles.

Varios fabricantes ofrecen servidores de vídeo bajo demanda. Los sistemas actuales están basados en los sistemas de archivos, ya que los sistemas de bases de datos existentes no proporcionan la respuesta en tiempo real que necesitan estas aplicaciones. La arquitectura básica de un sistema de vídeo bajo demanda comprende:

- **Servidor de vídeo.** Los datos multimedia se almacenan en varios discos (generalmente en una configuración RAID). Puede que los sistemas que contienen un gran volumen de datos utilicen medios de almacenamiento terciario para los datos a los que se tiene acceso con menor frecuencia.
- **Terminales.** La gente ve los datos multimedia mediante varios dispositivos, colectivamente denominados *terminales*. Ejemplos son las computadoras personales y los televisores conectados a una computadora pequeña y de coste reducido denominada microcomputadora.
- **Red.** La transmisión de los datos multimedia desde el servidor hasta los terminales necesita una red de gran capacidad.

El servicio de vídeo bajo demanda acabará siendo ubicuo, igual que lo son ahora la televisión por ondas hercianas y la televisión por cable. En el momento actual las aplicaciones principales de la tecnología de los videoservidores están en las oficinas (para formación, visión de conferencias y presentaciones grabadas y similares), en los hoteles y en las instalaciones de producción de vídeo.

23.4.3. Recuperación basada en la semejanza

En muchas aplicaciones multimedia los datos sólo se describen en la base de datos de manera aproximada. Un ejemplo son los datos de huellas dactilares del Apartado 23.4. Otros ejemplos son:

- **Datos gráficos.** Dos gráficos o imágenes que sean ligeramente diferentes en su representación en la base de datos pueden ser considerados iguales por un usuario. Por ejemplo, una base de datos puede almacenar diseños de marcas comerciales. Cuando haya que registrar una nueva marca puede que el sistema necesite identificar antes todas las marcas parecidas que se registraron anteriormente.
- **Datos de sonido.** Se están desarrollando interfaces de usuario basadas en el reconocimiento de la voz que permiten a los usuarios dar un comando o identificar un elemento de datos por la voz. Debe

comprobarse la semejanza de la entrada del usuario con los comandos o los elementos de datos almacenados en el sistema.

- **Datos manuscritos.** La entrada manuscrita puede utilizarse para identificar un elemento de datos manuscrito o una orden manuscrita almacenados en la base de datos. Una vez más, se necesita comprobar la semejanza.

El concepto de semejanza suele ser subjetivo y específico del usuario. No obstante, la verificación de la semejanza suele tener más éxito que el reconocimiento de voz o de letras manuscritas, ya que la entrada puede compararse con datos que ya se hallan en el sistema y, por tanto, el conjunto de opciones disponibles para el sistema es limitado.

Hay varios algoritmos para hallar las mejores coincidencias con una entrada dada mediante la comprobación de la semejanza. Algunos sistemas, incluidos un sistema telefónico de llamada por nombre activado por la voz, se han distribuido comercialmente. Véanse las notas bibliográficas para hallar más referencias.

23.5. COMPUTADORAS PORTÁTILES Y BASES DE DATOS PERSONALES

Las bases de datos comerciales de gran tamaño se han almacenado tradicionalmente en las instalaciones informáticas centrales. En las aplicaciones de bases de datos distribuidas ha habido generalmente una fuerte administración central de las bases de datos y de la red. Dos tendencias tecnológicas se han combinado para crear aplicaciones en las cuales la suposición de un control y de una administración centrales no es completamente correcta:

1. El uso cada vez más extendido de computadoras personales y, sobre todo, de computadoras portátiles.
2. El desarrollo de una infraestructura inalámbrica de comunicaciones digitales de coste relativamente bajo, basada en redes inalámbricas de área local, redes celulares de paquetes digitales y otras tecnologías.

La **informática móvil** se ha demostrado útil en muchas aplicaciones. Muchos profesionales viajeros utilizan computadoras portátiles para poder trabajar y tener acceso a los datos durante el viaje. Los servicios de mensajería utilizan computadoras portátiles para ayudar al seguimiento de los paquetes. Los servicios de emergencia utilizan computadoras portátiles en el escenario de los desastres, en las emergencias médicas y similares para tener acceso a la información y para introducir datos relativos a la situación. Siguen surgiendo nuevas aplicaciones de las computadoras portátiles.

Las computadoras comunicadas por radio crean una situación en que las máquinas ya no tienen ubicaciones fijas ni direcciones de red. Las **consultas dependientes de la ubicación** son una clase interesante de consultas que está motivada por las computadoras portátiles; en estas consultas la ubicación del usuario (computadora) es un parámetro de la consulta. El valor del parámetro de ubicación lo proporciona el usuario o, cada vez más, un sistema de posicionamiento global (GPS). Un ejemplo son los sistemas de información para viajeros que proporcionan a los conductores datos sobre los hoteles, los servicios de carretera y similares. El procesamiento de las consultas sobre los servicios que se hallan más adelante en la ruta actual debe basarse en la ubicación del usuario, en su dirección de movimiento y en su velocidad. Se ofrecen cada vez más ayudas a la navegación como una característica integrada en los automóviles.

La energía (la carga de las baterías) es un recurso escaso para la mayor parte de las computadoras portátiles. Esta limitación influye en muchos aspectos del diseño de los sistemas. Entre las consecuencias más interesantes de la necesidad de eficiencia energética está el empleo de emisiones programadas de datos para reducir la necesidad de transmisión de consultas de los sistemas portátiles.

Cantidades cada vez mayores de datos residen en máquinas administradas por los usuarios en lugar de por administradores de bases de datos. Además, estas máquinas pueden estar, a veces, desconectadas de la red. En muchos casos hay un conflicto entre la necesidad del usuario de seguir trabajando mientras está desconectado y la necesidad de consistencia global de los datos. En los apartados 23.5.1 a 23.5.4 se estudian técnicas en uso y en desarrollo para tratar los problemas de las computadoras portátiles y de la informática personal.

23.5.1. Un modelo de informática móvil

El entorno de la informática móvil consiste en computadoras portátiles, denominadas **anfitriones móviles**, y una red de computadoras conectadas por cable. Los anfitriones móviles se comunican con la red de cable mediante computadoras denominadas **estaciones para el soporte de movilidad**. Cada estación para el soporte de movilidad gestiona los anfitriones móviles de su **celda**, es decir, del área geográfica que cubre. Los anfitriones móviles pueden moverse de unas celdas a otras, por lo que necesitan el **relevo** del control de una estación para el soporte de movilidad a otra. Dado que los anfitriones móviles pueden, a veces, estar apagados, un anfitrión puede abandonar una celda y aparecer más tar-

de en otra distante. Por tanto, los movimientos de unas celdas a otras no se realizan necesariamente entre celdas adyacentes. Dentro de un área pequeña, como un edificio, los anfitriones móviles pueden conectarse mediante una red inalámbrica de área local que proporciona conectividad de coste más reducido que las redes celulares de área amplia, y que reduce la sobrecarga de entregas.

Es posible que los anfitriones móviles se comuniquen directamente sin intervención de ninguna estación para el soporte de movilidad. No obstante, esa comunicación sólo puede ocurrir entre anfitriones cercanos. Estas formas directas de comunicación se están haciendo más frecuentes con la llegada del estándar **Bluetooth**. Bluetooth utiliza radio digital de corto alcance para permitir la conectividad por radio a alta velocidad (hasta 721 kilobits por segundo) a distancias inferiores a diez metros. Concebido inicialmente como una sustitución de los cables, lo más prometedor de Bluetooth es la conexión ad hoc sencilla entre computadoras portátiles, PDAs, teléfonos celulares y las denominadas aplicaciones inteligentes.

La infraestructura de red para la informática móvil consiste en gran parte en dos tecnologías: redes inalámbricas locales (como la red de área local Orinoco de Avaya) y redes de telefonía celular basadas en paquetes. Los primeros sistemas celulares utilizaban tecnología analógica y estaban diseñados para la comunicación de voz. Los sistemas digitales de segunda generación siguieron centrándose en las aplicaciones de voz. Los sistemas de tercera generación (3G) y los denominados sistemas 2.5G utilizan redes basadas en paquetes y están más adaptados a las aplicaciones de datos. En estas redes la voz es sólo una más de las aplicaciones (aunque una económicamente importante).

Bluetooth, las redes de área local inalámbricas y las redes celulares 2.5G y 3G hacen posible que se comuniquen a bajo coste gran variedad de dispositivos. Aunque esta comunicación en sí misma no encaja en el dominio de las aplicaciones habituales de bases de datos, los datos de la contabilidad, del control y de la administración correspondientes a esta comunicación generan bases de datos enormes. La inmediatez de la comunicación por radio genera la necesidad de acceso en tiempo real a muchas de estas bases de datos. Esta necesidad de inmediatez añade otra dimensión a las restricciones del sistema, un asunto que se abordará en profundidad en el Apartado 24.3.

El tamaño y las limitaciones de potencia de muchas computadoras portátiles han llevado a la creación de jerarquías de memoria alternativas. En lugar de, o además de, el almacenamiento en disco, puede incluirse la memoria flash, que se estudió en el Apartado 11.1. Si el anfitrión móvil incluye un disco duro, puede que se permita que el disco deje de girar cuando no se utilice, para ahorrar energía. Las mismas consideraciones de tamaño y de energía limitan el tipo y el tamaño de las pantallas utilizadas en los dispositivos portátiles. Los diseñadores de los dispositivos portátiles suelen crear interfaces de usuario especiales para trabajar con estas restricciones. No obstante, la necesidad de presentar datos basados en la web ha exigido la creación de estándares para presentaciones. El **protocolo de aplicaciones inalámbrico** (Wireless Application Protocol, WAP) es un estándar para el acceso inalámbrico a internet. Los exploradores basados en WAP tienen acceso a páginas web especiales que utilizan el **lenguaje de marcas inalámbrico** (Wireless Markup Language**,** WML), un lenguaje basado en XML diseñado para las restricciones de la exploración web móvil e inalámbrica.

23.5.2. Encaminamiento y procesamiento de consultas

La ruta entre cada par de anfitriones puede cambiar con el tiempo si alguno de los dos anfitriones es móvil. Este sencillo hecho tiene un efecto espectacular en el nivel de la red, puesto que las direcciones de red basadas en las ubicaciones ya no son constantes en el sistema.

La informática móvil también afecta directamente al procesamiento de consultas de las bases de datos. Como se vio en el Capítulo 19, hay que considerar los costes de comunicación cuando se escoge una estrategia de procesamiento distribuido de las consultas. La informática móvil hace que los costes de comunicación cambien de manera dinámica, lo que complica el proceso de optimización. Además, hay varios conceptos de coste que se deben considerar en relación con los demás:

- El **tiempo del usuario** es una materia prima muy valiosa en muchas aplicaciones profesionales.
- El **tiempo de conexión** es la unidad por la que se asignan los costes monetarios en algunos sistemas de telefonía celular.
- El **número de bytes, o de paquetes, transferidos** es la unidad por la que se calculan los costes en algunos sistemas de telefonía celular digital.
- Los **costes basados en la hora del día** varían, en función de si la comunicación se produce durante los periodos pico o durante los periodos valle.
- La **energía** es limitada. A menudo la energía de las baterías es un recurso escaso cuyo uso debe optimizarse. Un principio básico de las comunicaciones inalámbricas es que hace falta menos energía para recibir señales de radio que para emitirlas. Así, la transmisión y la recepción de los datos imponen demandas de energía diferentes al anfitrión móvil.

23.5.3. Datos de difusión

Suele ser deseable para los datos que se solicitan con frecuencia que los transmitan las estaciones de soporte de las computadoras portátiles en un ciclo continuo, en lugar de que se transmitan a los anfitriones móviles a petición de éstos. Una aplicación típica de estos **datos**

de difusión es la información de las cotizaciones bursátiles. Hay dos motivos para utilizar los datos de difusión. En primer lugar, los anfitriones móviles evitan el coste energético de transmitir las solicitudes de datos. En segundo lugar, los datos de difusión pueden recibirlos simultáneamente gran número de anfitriones móviles, sin coste adicional. Por tanto, el bando de ancha disponible para transmisiones se utiliza de manera más efectiva.

Así, los anfitriones móviles pueden recibir los datos a medida que se transmiten, en lugar de consumir energía transmitiendo solicitudes. Puede que los anfitriones móviles tengan almacenamiento no volátil disponible para guardar en la caché los datos de difusión para su empleo posterior. Dada una consulta, los anfitriones móviles pueden minimizar los costes energéticos determinando si pueden procesarla sólo con los datos guardados en la caché. Si los datos guardados en la caché son insuficientes, hay dos opciones: Esperar a que los datos se transmitan o transmitir una solicitud de datos. Para tomar esta decisión los anfitriones móviles deben conocer el momento en que se transmitirán los datos en cuestión.

Los datos de difusión pueden transmitirse de acuerdo con una programación fija o según una programación variable. En el primer caso, los anfitriones móviles utilizan la programación fija conocida para determinar el momento en que se transmitirán los datos en cuestión. En el segundo caso, se debe transmitir la propia programación de transmisiones en una frecuencia de radio conocida y a intervalos de tiempos conocidos.

En efecto, el medio transmitido puede modelarse como un disco con una latencia elevada. Las solicitudes de datos pueden considerarse atendidas cuando los datos solicitados se transmiten. Las programaciones de transmisión se comportan como los índices de los discos. Las notas bibliográficas citan trabajos de investigación recientes en el área de administración de los datos de difusión.

23.5.4. Desconexiones y consistencia

Dado que puede que las comunicaciones inalámbricas se paguen con arreglo al tiempo de conexión, hay un incentivo para que se desconecten determinados anfitriones móviles durante periodos de tiempo considerables. Las computadoras portátiles sin conectividad inalámbrica están desconectadas la mayor parte del tiempo en que se utilizan, excepto cuando se conectan de manera periódica a sus computadoras anfitrionas, físicamente o mediante una red informática.

Durante esos periodos de desconexión puede que el anfitrión móvil siga operativo. Puede que el usuario del anfitrión móvil formule consultas y solicite actualizaciones de los datos que residen localmente o que se han guardado en la caché local. Esta situación crea varios problemas, en especial:

- **Recuperabilidad:** Las actualizaciones introducidas en una máquina desconectada pueden perderse si el anfitrión móvil sufre un fallo catastrófico. Dado que el anfitrión móvil representa un único punto de fallo, no se puede simular bien el almacenamiento estable.
- **Consistencia:** Los datos guardados en la caché local pueden quedar obsoletos, pero el anfitrión móvil no puede descubrir la situación hasta que vuelva a conectarse. De manera parecida, las actualizaciones que se produzcan en el anfitrión móvil no pueden transmitirse hasta que se produzca la reconexión.

El problema de la consistencia se exploró en el Capítulo 19, donde se estudiaron las particiones de la red, y aquí se partirá de esa base. En los sistemas distribuidos conectados por redes físicas las particiones se consideran un modo de fallo; en la informática móvil las particiones mediante desconexiones son parte del modo de operación normal. Por tanto, es necesario permitir que continúe el acceso a los datos a pesar de las particiones, pese al riesgo de que se produzca una pérdida de consistencia.

Para los datos actualizados sólo por el anfitrión móvil, es sencillo transmitir las actualizaciones cuando el anfitrión móvil vuelve a conectarse. No obstante, si el anfitrión móvil guarda en la caché copias de los datos sólo para lectura que pueden actualizar otras computadoras, puede que los datos guardados en la caché acaben siendo inconsistentes. Cuando se conecta el anfitrión móvil, puede recibir **informes de invalidación** que lo informen de las entradas de la caché que están obsoletas. No obstante, cuando el anfitrión móvil esté desconectado puede perder algún informe de invalidación. Una solución sencilla a este problema es invalidar toda la caché al volver a conectar el anfitrión móvil, pero una solución tan extrema resulta muy costosa. En las notas bibliográficas se citan varios esquemas para el almacenamiento en la caché.

Si se pueden producir actualizaciones tanto en el anfitrión móvil como en el resto del sistema, la detección de las actualizaciones conflictivas resulta más difícil. Los esquemas basados en la **numeración de versiones** permiten las actualizaciones de los archivos compartidos desde los anfitriones desconectados. Estos esquemas no garantizan que las actualizaciones sean consistentes. Más bien, garantizan que, si dos anfitriones actualizan de manera independiente la misma versión del documento, el conflicto se acabará descubriendo, cuando los anfitriones intercambien información directamente o mediante un anfitrión común.

El **esquema del vector de versiones** detecta las inconsistencias cuando las copias de un documento se actualizan de manera independiente. Este esquema permite que las copias de un *documento* se almacenen en varios anfitriones. Aunque se utilice el término *documento*, el esquema puede aplicarse a otros elementos de datos, como las tuplas de una relación.

La idea básica es que cada anfitrión i almacene, con su copia de cada documento d, un **vector de versiones,** es decir, un conjunto de números de versiones $\{V_{d,i}[j]\}$, con una entrada para cada uno de los demás anfitriones j en los que se puede actualizar potencialmente el documento. Cuando un anfitrión i actualiza un documento d, incrementa el número de versión $V_{d,i}[i]$ en una unidad.

Siempre que dos anfitriones i y j se conectan entre sí, intercambian los documentos actualizados, de modo que los dos obtienen versiones nuevas de los documentos. No obstante, antes de intercambiar los documentos, los anfitriones tienen que averiguar si las copias son consistentes:

1. Si los vectores de versiones son iguales en los dos anfitriones —es decir, si para cada k, $V_{d,i}[k] = V_{d,j}[k]$— las copias del documento d son idénticas.
2. Si, para cada k, $V_{d,i}[k] \leq V_{d,j}[k]$ y los vectores de versiones no son idénticos, la copia del documento d del anfitrión i es más antigua que la del anfitrión j. Es decir, la copia del documento d del anfitrión j se obtuvo mediante una o más modificaciones de la copia del documento del anfitrión i. El anfitrión i sustituye su copia de d, así como su copia del vector de versiones de d, por las copias del anfitrión j.
3. Si hay un par de anfitriones k y m tales que $V_{d,i}[k] < V_{d,j}[k]$ y $V_{d,i}[m] > V_{d,j}[m]$, las copias son *inconsistentes*; es decir, la copia de d de i contiene actualizaciones realizadas por el anfitrión k que no se han transmitido al anfitrión j y, de manera parecida, la copia de d de j contiene actualizaciones llevadas a cabo por el anfitrión m que no se han transmitido al anfitrión i. Entonces, las copias de d son inconsistentes, ya que se han realizado de manera independiente dos o más actualizaciones de d. Puede que se necesite la intervención manual para mezclar las actualizaciones.

El esquema de vectores de versiones se diseñó inicialmente para tratar los fallos en los sistemas de archivos distribuidos. El esquema adquirió mayor importancia porque las computadoras portátiles suelen almacenar copias de los archivos que también se hallan presentes en los sistemas servidores, lo que constituye de facto un sistema de archivos distribuido que suele estar desconectado. Otra aplicaciones de este esquema son los sistemas en grupo, en que los anfitriones se conectan de manera periódica, en lugar de hacerlo de manera continua, y deben intercambiar los documentos actualizados. El esquema del vector de versiones también tiene aplicaciones en las bases de datos replicadas.

No obstante, el esquema del vector de versiones no logra abordar el problema más difícil e importante que plantean las actualizaciones de los datos compartidos, la reconciliación de las copias inconsistentes de los datos. Muchas aplicaciones pueden llevar a cabo de manera automática la reconciliación ejecutando en cada computadora las operaciones que han conducido a las actualizaciones en las computadoras remotas durante el periodo de desconexión. Esta solución funciona si las operaciones de actualización conmutan, es decir, generan el mismo resultado, independientemente del orden en que se ejecuten. Puede que se disponga de técnicas alternativas en ciertas aplicaciones; en el peor de los casos, no obstante, debe dejarse a los usuarios que resuelvan las inconsistencias. El tratamiento automático de estas inconsistencias y la ayuda a los usuarios para que resuelvan las que no puedan tratarse de manera automática sigue siendo un área de investigación.

Otra debilidad es que el esquema del vector de versiones exige una comunicación sustancial entre el anfitrión móvil que vuelve a conectarse y su estación para el soporte de movilidad. Las verificaciones de la consistencia pueden posponerse hasta que se necesiten los datos, aunque este retraso puede incrementar la inconsistencia global de la base de datos.

La posibilidad de desconexión y el coste de las comunicaciones inalámbricas limitan el aspecto práctico de las técnicas de procesamiento de las transacciones para los sistemas distribuidos estudiadas en el Capítulo 19. A menudo resulta preferible dejar que los usuarios preparen las transacciones en los anfitriones móviles y exigir que, en lugar de ejecutarlas localmente, las remitan al servidor para su ejecución. Las transacciones que afectan a más de una computadora y que incluyen un anfitrión móvil afrontan bloqueos de larga duración durante el compromiso de la transacción, a menos que las desconexiones sean raras o predecibles.

23.6. RESUMEN

- El tiempo desempeña un papel importante en los sistemas de bases de datos. Las bases de datos son modelos del mundo real. Aunque la mayor parte de las bases de datos modelan el estado del mundo real en un momento dado (en el momento actual), las bases de datos temporales modelan los estados del mundo real a lo largo del tiempo.
- Los hechos de las relaciones temporales tienen momentos asociados cuando son válidos, que pueden representarse como una unión de intervalos. Los lenguajes de consulta temporales simplifican el modelado del tiempo, así como las consultas relacionadas con el tiempo.

- Las bases de datos espaciales se utilizan cada vez más hoy en día para almacenar datos de diseño asistido por computadora y datos geográficos.
- Los datos de diseño se almacenan sobre todo como datos vectoriales; los datos geográficos consisten en una combinación de datos vectoriales y lineales. Las restricciones de integridad espacial son importantes para los datos de diseño.
- Los datos vectoriales pueden codificarse como datos de la primera forma normal o almacenarse mediante estructuras que no sean la primera forma normal, como las listas. Las estructuras de índices de finalidad espacial resultan especialmente importantes para tener acceso a los datos espaciales y para procesar las consultas espaciales.
- Los árboles R son una extensión multidimensional de los árboles B; con variantes como los árboles R^+ y los árboles R*, se han hecho populares en las bases de datos espaciales. Las estructuras de índices que dividen el espacio de manera regular, como los árboles cuadráticos, ayudan a procesar las consultas de mezcla espaciales.
- Las bases de datos multimedia están aumentando de importancia. Problemas como la recuperación basada en la semejanza y la entrega de datos a velocidades garantizadas son temas de investigación actuales.
- Los sistemas de informática móvil se han vuelto de uso común, lo que ha llevado al interés por los sistemas de bases de datos que pueden ejecutarse en ellos. El procesamiento de las consultas en estos sistemas puede implicar la búsqueda en las bases de datos de los servidores. El modelo de coste de las consultas debe contener el coste de la comunicación, incluido el coste monetario y el coste de la energía de las baterías, que resulta relativamente elevado para los sistemas portátiles.
- La transmisión resulta mucho más económica por receptor que la comunicación punto a punto, y la transmisión de datos como los datos bursátiles ayuda a los sistemas portátiles a recoger los datos de manera económica.
- La operación en desconexión, el empleo de los datos de difusión y el almacenamiento de los datos en la caché son tres problemas importantes que se están abordando hoy en día en la informática móvil.

TÉRMINOS DE REPASO

- Árboles cuadráticos
  - Árbol cuadrático PR
  - Árbol cuadrático regional
- Árboles k-d
- Árboles k-d-B
- Árboles R
  - Caja límite
  - División cuadrática
- Bases de datos de diseño
- Bases de datos multimedia
- Consistencia
  - Esquema del vector de versiones
  - Informes de invalidación
- Consultas dependientes de la ubicación
- Consultas espaciales
- Consultas de proximidad
- Consultas regionales
- Consultas de vecino más próximo
- Datos de difusión
- Datos de diseño asistido por computadora (Computer-Aided-Design, CAD)
- Datos espaciales y geográficos
- Datos geográficos
- Datos isócronos
- Datos por líneas (raster)
- Datos de medios continuos
- Datos temporales
- Datos vectoriales
- Formatos de datos multimedia
- Indexado de los datos espaciales
- Informática móvil
  - Anfitriones móviles
  - Celda
  - Estaciones de soporte de las computadoras portátiles
  - Relevo
- Lenguajes de consulta temporales
- Mezcla temporal
- Proyección temporal
- Recuperación basada en la semejanza
- Relación bitemporal
- Relación instantánea
- Relación temporal
- Reunión espacial
- Selección temporal
- Servidores de vídeo
- Sistemas de información geográfica
- Sistema de posicionamiento global (Global Positioning System, GPS)
- Tiempo de transacción
- Tiempo universal coordinado (UTC)
- Tiempo válido
- Triangulación

EJERCICIOS

23.1. Indíquense los dos tipos de tiempo y en lo que se diferencian. Indíquese el motivo de que haya dos tipos de tiempo asociados con cada tupla.

23.2. Indíquese si se conservarán las dependencias funcionales si se convierte una relación en una relación temporal añadiéndole un atributo temporal. Indíquese el modo en que se resuelve el problema en las bases de datos temporales.

23.3. Supóngase que se tiene una relación que contiene las coordenadas *x, y* y los nombres de varios restaurantes. Supóngase también que las únicas consultas que se formularán serán de la forma siguiente: La consulta especifica un punto y pregunta si hay algún restaurante exactamente en ese punto. Indíquese el tipo de índice que sería preferible, árbol R o árbol B. Indíquese el motivo.

23.4. Considérense dos datos vectoriales bidimensionales en que los elementos de datos no se solapan. Indíquese si es posible convertir esos datos vectoriales en datos lineales. En caso de que sea posible, indíquense los inconvenientes de almacenar los datos lineales obtenidos de esa conversión en lugar de los datos vectoriales originales.

23.5. Supóngase que se dispone de una base de datos espacial que soporta consultas regionales (con regiones circulares) pero no consultas de vecino más próximo. Descríbase un algoritmo para hallar el vecino más próximo haciendo uso de varias consultas regionales.

23.6. Supóngase que se desean almacenar segmentos rectilíneos en un árbol R. Si un segmento rectilíneo no es paralelo a los ejes, su caja límite puede ser grande y contener una gran área vacía.

- Descríbase el efecto en el rendimiento de tener cajas límite de gran tamaño en las consultas que piden los segmentos rectilíneos que intersectan una región dada.
- Descríbase brevemente una técnica para mejorar el rendimiento de esas consultas y dese un ejemplo de sus ventajas. Consejo: Se pueden dividir los segmentos en partes más pequeñas.

23.7. Dese un procedimiento recursivo para calcular de manera eficiente la mezcla espacial de dos relaciones con índices de árbol R. (Consejo: Utilícense cajas límite para comprobar si las entradas hojas bajo un par de nodos internos pueden intersectarse.)

23.8. Estúdiese el soporte de los datos espaciales ofrecido por el sistema de bases de datos que se está utilizando e impleméntese lo siguiente:

a. Un esquema para representar la ubicación geográfica de los restaurantes y características como la cocina que se sirve en cada restaurante y su nivel de precios.

b. Una consulta para hallar los restaurantes económicos que sirven comida india y que se hallan a menos de nueve kilómetros de casa del lector (supóngase cualquier ubicación para la casa del lector).

c. Una consulta para hallar para cada restaurante su distancia al restaurante más cercano que sirve la misma cocina y con el mismo nivel de precios.

23.9. Indíquense los problemas que se producen en un sistema de medios continuos si los datos se entregan demasiado lento o demasiado rápido.

23.10. Descríbase el modo en que las ideas subyacentes a la organización RAID (Apartado 11.3) pueden utilizarse en un entorno de datos de difusión, donde puede que haya ocasionalmente ruido que impida la recepción de parte de los datos que se están transmitiendo.

23.11. Indíquense tres características principales de la informática móvil en redes inalámbricas que son diferentes de las de los sistemas distribuidos tradicionales.

23.12. Indíquense tres factores que haya que considerar en la optimización de las consultas para la informática móvil que no se consideren en los optimizadores de consultas tradicionales.

23.13. Defínase un modelo en que se difundan repetidamente los datos en el que el medio de transmisión se modele como un disco virtual. Descríbase el modo en que el tiempo de acceso y la velocidad de transferencia de datos del disco virtual se diferencian de los valores correspondientes a un disco duro normal.

23.14. Considérese una base de datos de documentos en la que todos los documentos se conserven en una base de datos central. En las computadoras portátiles se guardan copias de algunos documentos. Supóngase que la computadora portátil A actualiza una copia del documento 1 mientras está desconectada y que, al mismo tiempo, la computadora portátil B actualiza una copia del documento 2 mientras está desconectada. Muéstrese el modo en que el esquema del vector versión puede asegurar la actualización adecuada de la base de datos central y de las computadoras portátiles cuando se vuelva a conectar una computadora portátil.

23.15. Dese un ejemplo para mostrar que el esquema del vector versión no asegura la secuenciabilidad. (Consejo: Utilícese el ejemplo del Ejercicio 23.14, con la suposición de que los documentos 1 y 2 están disponibles en las dos computadoras portátiles A y B, y téngase en cuenta la posibilidad de que un documento pueda leerse sin que se actualice.)

NOTAS BIBLIOGRÁFICAS

La incorporación del tiempo en el modelo relacional de datos se estudia en Snodgrass y Ahn [1985], Clifford y Tansel [1985], Gadia [1986], Gadia [1988], Snodgrass [1987], Tansel et al. [1993], Snodgrass et al. [1994] y Tuzhilin and Clifford [1990]. Stam y Snodgrass [1988] y Soo [1991] proporcionan estudios sobre la administración de los datos temporales. Jensen et al. [1994] presentan un glosario de conceptos de las bases de datos temporales, con la intención de unificar la terminología, un propósito que tuvo un impacto significativo en el estándar SQL. Tansel et al. [1993] es una colección de artículos sobre diferentes aspectos de las bases de datos temporales. Chomicki [1995] presenta técnicas para administrar las restricciones para la integridad temporal. Un concepto de completitud para los lenguajes de consultas temporales análogo a la completitud relacional (equivalencia con el álgebra relacional) se da en Clifford et al. [1994].

Samet [1995b] proporciona una introducción a la gran cantidad de trabajo realizado sobre las estructuras espaciales de índices. Samet [1990] proporciona una cobertura en el nivel de los libros de texto de las estructuras espaciales de datos. Una de las primeras descripciones de los árboles cuadráticos se proporciona en Finkel y Bentley [1974]. Samet [1990] y Samet [1995b] describen numerosas variantes de los árboles cuadráticos. Bentley [1975] describe los árboles k-d, y Robinson [1981] describe los árboles k-d-B. Los árboles R se presentaron originalmente en Guttman [1984]. Las extensiones de los árboles R se presentan en Sellis et al. [1987], que describen los árboles R^+; Beckmann et al. [1990], que describen el árbol R^*; y Kamel y Faloutsos [1992], que describen una versión paralela de los árboles R.

Brinkhoff et al. [1993] estudian una implementación de las mezclas espaciales mediante árboles R. Lo y Ravishankar [1996] y Patel y DeWitt [1996] presentan los métodos basados en las particiones para el cálculo de las mezclas espaciales. Samet y Aref [1995] proporcionan una introducción de los modelos espaciales de datos, de las operaciones espaciales y de la integración de los datos espaciales con los no espaciales. El indexado de los documentos manuscritos se estudia en Aref et al. [1995b], Aref et al. [1995a] y Lopresti y Tomkins [1993]. Las mezclas de los datos aproximados se estudian en Barbará et al. [1992]. Evangelidis et al. [1995] presentan una técnica para el acceso concurrente a los índices de los datos espaciales.

Samet [1995a] describe los campos de investigación en las bases de datos multimedia. El indexado de los datos multimedia se estudia en Faloutsos y Lin [1995]. Los servidores de vídeo se estudian en Anderson et al. [1992], Rangan et al. [1992], Ozden et al. [1994], Freedman y DeWitt [1995] y Ozden et al. [1996b]. La tolerancia a los fallos se estudia en Berson et al. [1995] y Ozden et al. [1996a]. Reason et al. [1996] sugieren esquemas alternativos de compresión para la transmisión de vídeo por redes inalámbricas. Las técnicas de administración del almacenamiento en disco para los datos de vídeo se describen en Chen et al. [1995], Chervenak et al. [1995], Ozden et al. [1995a] y Ozden et al. [1995b].

La administración de la información en los sistemas que incluyen computadoras portátiles se estudia en Alonso y Korth [1993] y en Imielinski y Badrinath [1994]. Imielinski y Korth [1996] presentan una introducción a la informática móvil y una colección de trabajos de investigación sobre el tema. El indexado de los datos de difusión por medios inalámbricos se estudia en Barbará e Imielinski [1994] y en Acharya et al. [1995]. La administración de los discos en las computadoras portátiles se aborda en Douglis et al. [1994].

El esquema del vector de versiones para la detección de la inconsistencia en los sistemas de archivos distribuidos se describe en Popek et al. [1981] y en Parker et al. [1983].

CAPÍTULO

24 PROCESAMIENTO AVANZADO DE TRANSACCIONES

En los Capítulos 15, 16 y 17 se introdujo el concepto de transacción, que es una unidad de programa que tiene acceso —y posiblemente actualiza— a varios elementos de datos, y cuya ejecución asegura la conservación de las propiedades ACID. En esos capítulos se estudiaron gran variedad de esquemas para asegurar las propiedades ACID en entornos en los que pueden producirse fallos, y en los que las transacciones pueden ejecutarse de manera concurrente.

En este capítulo se irá más allá de los esquemas básicos estudiados anteriormente y se abordarán los conceptos del procesamiento avanzado de las transacciones, incluidos los monitores de procesamiento de transacciones, los flujos de trabajo de las transacciones, las bases de datos en memoria principal, las bases de datos en tiempo real, las transacciones de larga duración, las transacciones anidadas y las transacciones con varias bases de datos.

24.1. MONITORES DE PROCESAMIENTO DE TRANSACCIONES

Los **monitores de procesamiento de transacciones** (**transaction-processing monitors, TP monitors**) son sistemas que se desarrollaron en los años setenta y ochenta del siglo pasado, inicialmente como respuesta a la necesidad de soportar gran número de terminales remotas (como los terminales de reserva de las líneas aéreas) desde una sola computadora. El término *TP monitor* significaba inicialmente *monitor de teleprocesamiento (Teleprocessing Monitor)*.

Los monitores TP han evolucionado desde entonces para ofrecer el soporte central para el procesamiento distribuido de las transacciones, y el término monitor TP ha adquirido su significado actual. El monitor CICS TP de IBM fue uno de los primeros monitores TP, y se ha utilizado mucho. Entre los monitores TP de la generación actual están Tuxedo y Top End (los dos actualmente de BEA Systems), Encina (de Transarc, que ahora forma parte de IBM) y Transaction Server (de Microsoft).

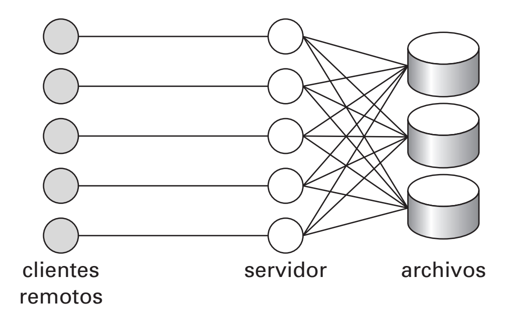
24.1.1. Arquitecturas de los monitores TP

Los sistemas de procesamiento de transacciones a gran escala se construyen en torno a una arquitectura cliente-servidor. Una manera de crear estos sistemas es tener un proceso servidor para cada cliente; el servidor realiza la autentificación, y luego ejecuta las acciones solicitadas por el cliente.

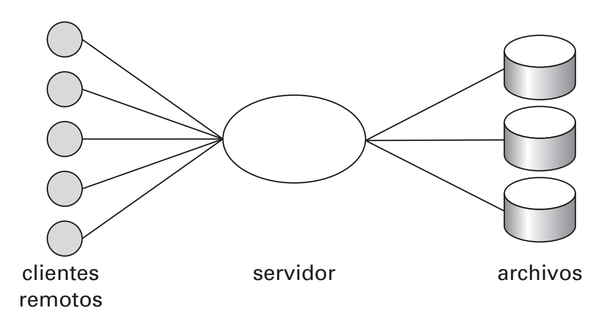
Este **modelo de proceso por cliente** se ilustra en la Figura 24.1. Este modelo presenta varios problemas con respecto a la utilización de la memoria y la velocidad de procesamiento:

- Los requisitos de memoria para cada proceso son elevados. Aunque se comparta la memoria para el código de los programas entre todos los procesos, cada proceso consume memoria para los datos locales y los descriptores de los archivos abiertos, así como para la sobrecarga del sistema operativo, como las tablas de páginas para soportar la memoria virtual.
- El sistema operativo divide el tiempo disponible de CPU entre los procesos conmutando entre ellos; esta tarea se denomina **multitarea**. Cada **cambio de contexto** entre un proceso y el siguiente supone una sobrecarga considerable de la CPU; incluso en los sistemas rápidos de hoy en día un cambio de contexto puede tardar cientos de microsegundos.

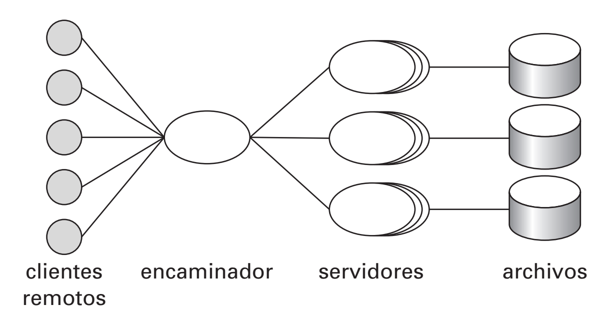
Los problemas anteriores pueden evitarse teniendo un proceso con un solo servidor al que se conecten todos los servidores; este modelo se denomina **modelo de servidor único**, ilustrado en la Figura 24.1b. Los clientes remotos envían las solicitudes al proceso del servidor, que ejecuta entonces esas solicitudes. Este modelo también se utiliza en los entornos cliente-servidor, en los que los clientes envían solicitudes a un proceso de un solo servidor. El proceso servidor asume las tareas, como la autentificación de los usuarios, que normalmente asumiría el sistema operativo. Para evitar bloquear otros clientes al procesar una solicitud de larga duración de un cliente, el servidor tiene **varias hebras**: El proceso servidor tiene una hebra de control para cada cliente y, en efecto, implementa su propia multitarea de baja sobrecarga. Ejecuta el código en nombre de un cliente duran-



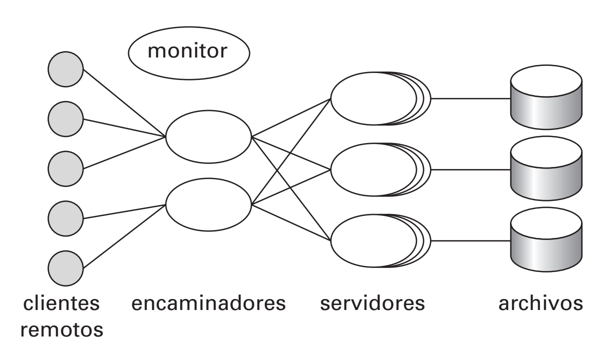
(a) Modelo de proceso por cliente



(b) Modelo de servidor único



(c) Modelo de muchos servidores y un solo encaminador



(d) Modelo de muchos servidores y muchos encaminadores

FIGURA 24.1. Arquitecturas de los monitores TP.

te un rato, luego guarda el contexto interno y conmuta al código de otro cliente. A diferencia de la sobrecarga de la multitarea, el coste de la conmutación entre hebras es reducido (generalmente sólo unos pocos microsegundos).

Los sistemas basados en el modelo de servidor único, como la versión original del monitor TP CICS de IBM y los servidores de archivos como NetWare de Novell, proporcionaban con éxito tasas de transacciones elevadas con recursos limitados. No obstante, tenían problemas, especialmente cuando varias aplicaciones tenían acceso a la misma base de datos:

- Dado que todas las aplicaciones se ejecutan como un único proceso, no hay protección entre ellas. Un fallo en una aplicación puede afectar también a todas las demás aplicaciones. Sería mejor ejecutar cada aplicación como un proceso separado.
- Estos sistemas no están adecuados a las bases de datos paralelas o distribuidas, ya que un proceso servidor no puede ejecutarse simultáneamente en varios servidores (sin embargo, las hebras concurrentes de un proceso pueden soportarse en un sistema multiprocesador de memoria compartida). Se trata de un inconveniente serio para las organizaciones de gran tamaño en las que el procesamiento paralelo resulta fundamental para el tratamiento de grandes cargas de trabajo, y los datos distribuidos son cada vez más frecuentes.

Una manera de resolver estos problemas es ejecutar varios procesos del servidor de aplicaciones que tengan acceso a una base de datos común y dejar que los clientes se comuniquen con la aplicación mediante un único proceso de comunicaciones que encamine las solicitudes. Este modelo se denomina **modelo de varios servidores y un solo encaminador**, ilustrado en la Figura 24.1c. Este modelo soporta procesos de servidor independientes para varias aplicaciones; además, cada aplicación puede tener un grupo de procesos de servidor, cualquiera de los cuales puede manejar una sesión cliente. La solicitud puede, por ejemplo, encaminarse al servidor con carga menor de un grupo. Como antes, cada proceso de servidor puede tener, a su vez, varias hebras, de modo que puede atender de manera concurrente varios clientes. Como generalización adicional, los servidores de aplicaciones pueden ejecutarse en sitios diferentes de una base de datos paralela o distribuida y el proceso de comunicaciones puede manejar las comunicaciones entre los procesos.

La arquitectura anterior también se utiliza mucho en los servidores web. Un servidor web tiene un proceso principal que recibe las solicitudes HTTP, y luego asigna la tarea de manejar cada solicitud a un proceso diferente (escogido de entre un grupo de procesos). Cada uno de los procesos tiene, a su vez, varias hebras, por lo que puede atender varias solicitudes.

Una arquitectura más general tiene varios procesos, en lugar de uno solo, para comunicarse con los clien-

tes. Los procesos de comunicación con los clientes interactúan con uno o varios procesos encaminadores, que encaminan las solicitudes hacia el servidor correspondiente. Los monitores TP de generaciones posteriores, por tanto, tienen una arquitectura diferente, denominada **modelo de varios servidores y varios encaminadores**, ilustrado en la Figura 24.1d. Un proceso controlador inicia los demás procesos y supervisa su funcionamiento. Pathway de Tandem es un ejemplo de los monitores TP de generaciones posteriores que utilizan esta arquitectura. Los sistemas servidores web de rendimiento muy elevado también adoptan una arquitectura de este tipo.

La estructura detallada de un monitor TP aparece en la Figura 24.2. Un monitor TP hace más cosas que pasar mensajes a los servidores de aplicaciones. Cuando llegan los mensajes, puede que haya que ubicarlos en una cola; por tanto, hay un **gestor de colas** para los mensajes entrantes. Puede que la cola sea una **cola duradera**, cuyas entradas sobreviven a los fallos del sistema. El empleo de colas duraderas ayuda a asegurar que se acaben procesando los mensajes una vez recibidos y guardados en la cola, independientemente de los fallos del sistema. La gestión de las autorizaciones y de los servidores de aplicaciones (por ejemplo, el inicio de los servidores y el encaminamiento de los mensajes hacia los servidores) son otras funciones de los monitores TP. Los monitores TP suelen proporcionar recursos para la elaboración de registros históricos, recuperación y control de concurrencia, lo que permite a los servidores de aplicaciones implementar directamente, si fuera necesario, las propiedades ACID de las transacciones.

Finalmente, los monitores TP también proporcionan soporte para la mensajería persistente. Hay que recordar que la mensajería persistente (Apartado 19.4.3) proporciona una garantía de que el mensaje se entregue si (y sólo si) la transacción se compromete.

Además de estos servicios, muchos monitores TP también proporcionaban *recursos para presentaciones* para crear interfaces de menús o de formularios o para los clientes no inteligentes como los terminales; estos recursos ya no son importantes porque los clientes no inteligentes ya no se utilizan mucho.

24.1.2. Coordinación de las aplicaciones mediante los monitores TP

Hoy en día las aplicaciones suelen tener que interactuar con varias bases de datos. Puede que tengan que interactuar con sistemas heredados, como los sistemas de almacenamiento de finalidad especial construidos directamente con base en los sistemas de archivos. Finalmente, puede que tengan que comunicarse con usuarios o con otras aplicaciones en sitios remotos. Por tanto, también tienen que interactuar con subsistemas de comunicaciones. Es importante poder coordinar los accesos a los datos e implementar las propiedades ACID de las propiedades a través de esos sistemas.

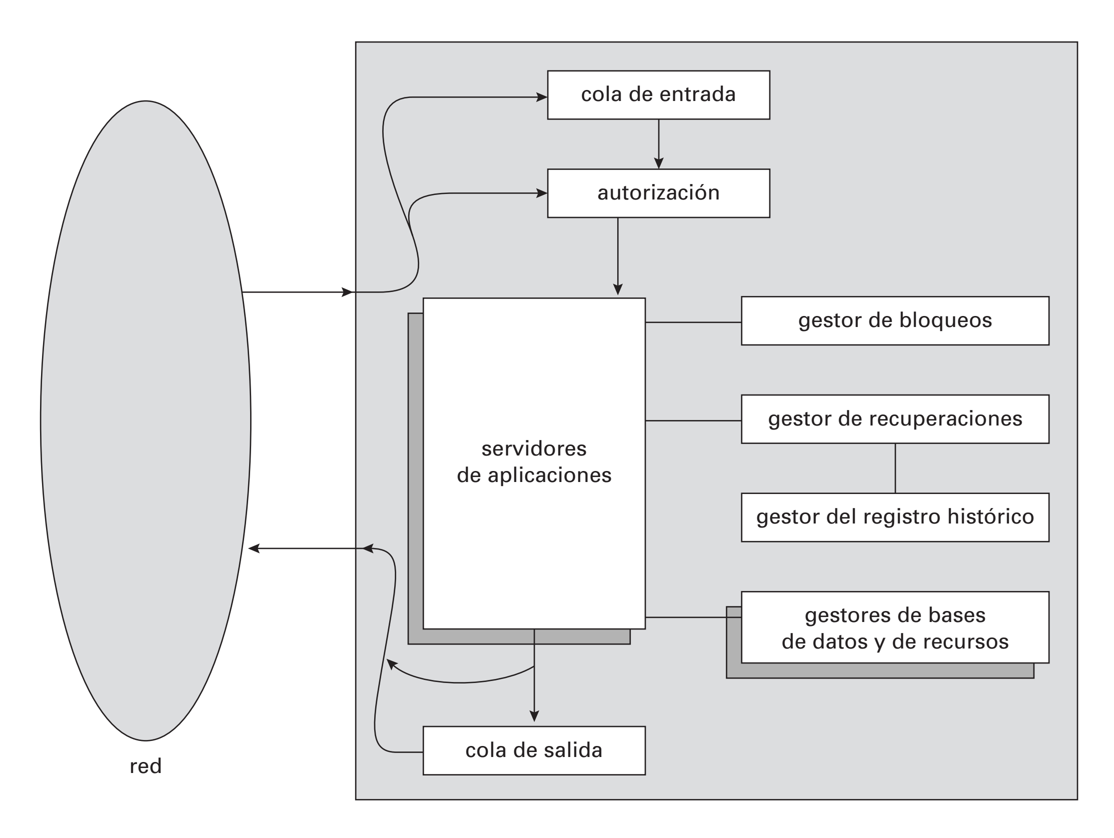


FIGURA 24.2. Componentes de los monitores TP.

Los monitores TP modernos proporcionan soporte para la construcción y la gestión de aplicaciones de un tamaño tan grande, creadas a partir de varios subsistemas como las bases de datos, los sistemas heredados y los sistemas de comunicaciones. Los monitores TP tratan cada subsistema como un **gestor de recursos** que proporciona acceso transaccional a algún conjunto de recursos. La interfaz entre el monitor TP y el gestor de recursos se define mediante un conjunto de primitivas de las transacciones como *begin_transacion* (iniciar transacción), *commit_transaction* (comprometer transacción), *abort_transaction* (abortar transacción) y *prepare_to_commit_transaction* (preparar para comprometer transacción, para el compromiso de dos fases). Por supuesto, el gestor de recursos también debe proporcionar otros servicios, como proporcionar datos, a la aplicación.

La interfaz del gestor de recursos está definida por el estándar de procesamiento de transacciones distribuidas X/Open. Muchos sistemas de bases de datos soportan los estándares X/Open, y pueden actuar como gestores de recursos. Los monitores TP —así como otros productos, como los sistemas SQL, que soportan los estándares X/Open— pueden conectarse con los gestores de recursos.

Además, los servicios proporcionados por los monitores TP, como la mensajería persistente y las colas duraderas, actúan como gestores de recursos que soportan las transacciones. Los monitores TP pueden actuar como coordinadores de los compromisos de dos fases para las transacciones que tienen acceso a estos servicios y a los sistemas de bases de datos. Por ejemplo, cuando se ejecuta una transacción de actualización encolada, se entrega un mensaje y se elimina la transacción solicitada de la cola de solicitudes. El compromiso de dos fases entre la base de datos y los gestores de recursos para las colas duraderas y para la mensajería persistente ayuda a asegurar que, independientemente de los fallos, pueden producirse todas estas acciones o ninguna de ellas.

También se pueden utilizar los monitores TP para administrar los sistemas complejos cliente-servidor que consisten en varios servidores y gran número de clientes. El monitor TP coordina las actividades como los puntos de control y los cierres del sistema. Proporciona la seguridad y la autentificación de los clientes. Administra los grupos de servidores añadiendo o eliminando servidores sin ninguna interrupción del sistema. Finalmente, controla el ámbito de los fallos. Si falla algún servidor, el monitor TP puede detectar ese fallo, abortar las transacciones en curso y reiniciarlas. Si falla algún nodo, el monitor TP puede migrar las transacciones a servidores de otros nodos y, una vez más, cancelar las transacciones incompletas. Cuando los nodos que fallan se reinician, el monitor TP puede gobernar la recuperación de los gestores de recursos del nodo.

Los monitores TP pueden utilizarse para ocultar fallos de las bases de datos en los sistemas replicados; los sistemas remotos de copia de seguridad (Apartado 17.10) son un ejemplo de sistemas replicados. Las solicitudes de transacciones se remiten al monitor TP, que transfiere los mensajes a una de las réplicas de la base de datos (al sitio principal, en el caso de sistemas remotos de copia de seguridad). Si falla algún sitio, el monitor TP puede encaminar los mensajes de manera transparente hacia un sitio de copia de seguridad, enmascarando el fallo del primer sitio.

En los sistemas cliente-servidor los clientes suelen interactuar con los servidores mediante un mecanismo de **llamada a procedimientos remotos (Remote-Procedure Call, RPC**), en el que el cliente realiza la llamada a un procedimiento, que se ejecuta realmente en el servidor, y los resultados se devuelven al cliente. En lo relativo al código cliente que invoca al RPC, la llamada tiene el mismo aspecto que la invocación a un procedimiento local. Los sistemas de monitores TP, como Encina, proporcionan una interfaz para **RPC transaccionales** con sus servicios. En esta interfaz el mecanismo RPC proporciona llamadas que pueden utilizarse para encerrar una serie de llamadas RPC dentro de una transacción. Por tanto, las actualizaciones llevadas a cabo por el RPC se ejecutan dentro del ámbito de la transacción y se pueden hacer retroceder si hay algún fallo.

24.2. FLUJOS DE TRABAJO DE TRANSACCIONES

Un **flujo de trabajo** es una actividad en la que varias entidades de procesamiento ejecutan varias tareas de manera coordinada. Una **tarea** define un trabajo que hay que hacer y puede especificarse de varias maneras, incluidos una descripción textual en un archivo o en un mensaje de correo electrónico, un formulario, un mensaje o un programa de computadora. La **entidad de procesamiento** que lleva a cabo las tareas puede ser una persona o un sistema de software (por ejemplo, un sistema de envío de correo electrónico, un programa de aplicación o un sistema gestor de bases de datos).

La Figura 24.3 muestra ejemplos de flujos de trabajo. Un ejemplo sencillo es el de un sistema de correo electrónico. La entrega de un solo mensaje de correo implica varios sistemas de envío de correo que reciben y transmiten el mensaje de correo, hasta que el mensaje alcance su destino, donde se almacena. Cada sistema de envío de correo lleva a cabo una tarea —transmitir el mensaje al siguiente sistema de envío de correo— y puede ser necesaria la tarea de varios sistemas de envío de correo para encaminar el mensaje desde su origen hasta su destino. Otros términos empleados en la lite-

Aplicación de flujo de trabajo	*Tarea típica*	*Entidad de procesamiento típica*
encaminamiento de correo electrónico	mensaje de correo electrónico	sistemas de envío de correo electrónico
procesamiento de préstamos	procesamiento de formularios	seres humanos, software de aplicaciones
procesamiento de órdenes de compra	procesamiento de formularios	seres humanos, software de aplicaciones, SGBD

FIGURA 24.3. Ejemplos de flujos de trabajo.

ratura de bases de datos y similares para hacer referencia a los flujos de trabajo son **flujo de tareas** y **aplicaciones multisistema**. Las tareas del flujo de trabajo a veces se denominan **pasos**.

En general, los flujos de trabajo pueden implicar a una o varias personas. Por ejemplo, considérese el procesamiento de un préstamo. El flujo de trabajo correspondiente aparece en la Figura 24.4. La persona que desea un préstamo rellena un formulario, que es revisado por el encargado de los préstamos. Un empleado que procesa las solicitudes de préstamos comprueba los datos del formulario, utilizando fuentes como las oficinas de referencia de préstamo. Cuando se ha reunido toda la información solicitada, el encargado de los préstamos puede que decida conceder el préstamo; puede que esa decisión tenga que ser aprobada por uno o más empleados de rango superior, después de lo cual se podrá conceder el préstamo. Cada persona de este flujo de trabajo realiza una tarea; en un banco que no tenga automatizada la tarea de procesamiento de los préstamos, la coordinación de las tareas suele ejecutarse pasando la solicitud del préstamo con notas y otra información adjuntas de un empleado al siguiente. Otros ejemplos de flujos de trabajo son el procesamiento de notas de gastos, de órdenes de compra y de transacciones de tarjetas de préstamo.

Hoy en día es más probable que toda la información relativa a un flujo de trabajo se almacene en forma digital en una o más computadoras y, con el auge de las redes, la información puede transferirse con facilidad de una computadora a otra. Por tanto, es viable que las organizaciones automaticen sus flujos de trabajo. Por ejemplo, para automatizar las tareas implicadas en el procesamiento de los préstamos, se puede almacenar la solicitud de préstamo y la información asociada en una base de datos. El propio flujo de trabajo implica, entonces, la transferencia de la responsabilidad de una persona a la siguiente y, posiblemente, incluso a programas que pueden capturar de manera automática la información necesaria. Las personas implicadas pueden coordinar sus actividades mediante el correo electrónico, por ejemplo.

Hay que abordar dos actividades, en general, para automatizar un flujo de trabajo. La primera es la **especificación del flujo de trabajo**: detallar las tareas que hay que ejecutar y definir los requisitos de la ejecución. El segundo problema es la **ejecución del flujo de trabajo**, que hay que llevar a cabo mientras se proporcionan las salvaguardas de los sistemas tradicionales de bases de datos relativas a corrección de los cálculos e integridad y durabilidad de los datos. Por ejemplo, no resulta aceptable que se pierda una solicitud de préstamo o una nota, ni que se procese más de una vez, debido a un fallo del sistema. La idea subyacente a los flujos de trabajo transaccionales es utilizar y ampliar los conceptos de las transacciones al contexto de los flujos de trabajo.

Las dos actividades se complican por el hecho de que muchas organizaciones utilizan varios sistemas de procesamiento de la información administrados de manera independiente que, en la mayor parte de los casos, se desarrollaron por separado para automatizar funciones diferentes. Puede que las actividades del flujo de trabajo exijan interacciones entre varios de esos sistemas, cada uno de las cuales lleva a cabo una tarea, así como interacciones con las personas.

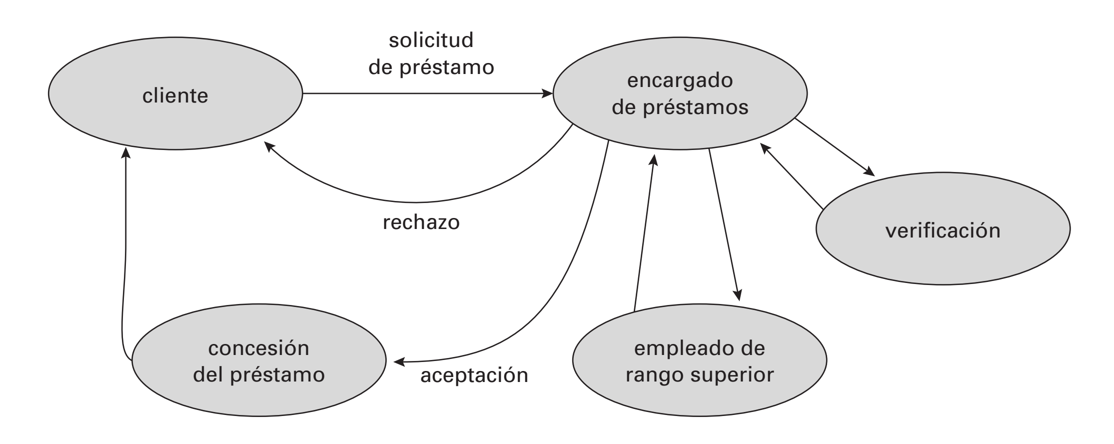


FIGURA 24.4. Flujo de trabajo en el procesamiento de préstamos.

En los últimos años se han desarrollado varios sistemas de flujo de trabajo. Aquí se estudiarán las propiedades de los sistemas de flujo de trabajo en un nivel relativamente abstracto, sin descender a los detalles de ningún sistema concreto.

24.2.1. Especificación del flujo de trabajo

No hace falta modelar los aspectos internos de cada tarea con vistas a la especificación y gestión de un flujo de trabajo. En una vista abstracta de las tareas, cada tarea puede utilizar los parámetros almacenados en sus variables de entrada, recuperar y actualizar los datos del sistema local, almacenar los resultados en sus variables de salida y se la puede consultar sobre su estado de ejecución. En cualquier momento de la ejecución el **estado del flujo de trabajo** consiste en el conjunto de estados de las tareas constituyentes del flujo de trabajo, y los estados (valores) de todas las variables de la especificación del flujo de trabajo.

La coordinación de las tareas puede especificarse de manera estadística o dinámica. La especificación estática define las tareas —y las dependencias entre ellas— antes de que comience la ejecución del flujo de trabajo. Por ejemplo, las tareas del flujo de trabajo de las notas de gastos pueden consistir en la aprobación de las notas por una secretaria, un gestor y un contable, en ese orden, y, finalmente, en la entrega de un cheque. Las dependencias entre las tareas puede ser sencilla —hay que completar cada tarea antes de que comience la siguiente.

Una generalización de esta estrategia es la imposición de una condición previa a la ejecución de cada tarea del flujo de trabajo, de modo que todas las tareas posibles del flujo de trabajo y sus dependencias se conozcan por anticipado, pero que sólo se ejecuten aquellas tareas cuyas condiciones previas se satisfagan. Las condiciones previas pueden definirse mediante dependencias como las siguientes:

- **El estado de ejecución** de otras tareas, por ejemplo, «la tarea t_i no puede comenzar hasta que la tarea t_j haya finalizado», o «la tarea t_i debe abortarse si la tarea t_j se ha comprometido».
- **Los resultados** de otras tareas, por ejemplo, «la tarea t_i puede comenzar si la tarea t_j devuelve un valor mayor que veinticinco», o «la tarea de aprobación por el gestor puede comenzar si la tarea de aprobación por la secretaria devuelve el resultado de Aceptar».
- **Las variables externas** modificadas por los eventos externos, por ejemplo, «la tarea t_i no puede iniciarse antes de las nueve de la mañana», o « la tarea t_i debe iniciarse antes de que transcurran veinticuatro horas desde la finalización de la tarea t_j».

Las dependencias pueden combinarse mediante los conectores lógicos (**or**, **and**, **not**) para formar condiciones previas complejas de planificación.

Un ejemplo de la planificación dinámica de las tareas son los sistemas de encaminamiento del correo electrónico. La tarea que hay que programar a continuación para cada mensaje de correo depende de la dirección de destino de ese mensaje y de los encaminadores intermedios que se hallan en funcionamiento.

24.2.2. Requisitos de atomicidad ante fallos de los flujos de trabajo

El diseñador del flujo de trabajo puede especificar los requisitos de **atomicidad ante fallos** del flujo de trabajo de acuerdo con la semántica del flujo de trabajo. El concepto tradicional de atomicidad ante fallos exige que el fallo de cualquier tarea dé lugar al fallo del flujo de trabajo. Sin embargo, un flujo de trabajo puede, en muchos casos, sobrevivir al fallo de una de sus tareas, por ejemplo, ejecutando una tarea funcionalmente equivalente en otro sitio. Por consiguiente, se debe permitir al diseñador que defina los requisitos de atomicidad ante fallos del flujo de trabajo. El sistema debe garantizar que cada ejecución de un flujo de trabajo termine en un estado que satisfaga los requisitos de atomicidad ante fallos definidos por el diseñador. Esos estados se denominan **estados aceptables de terminación** del flujo de trabajo. Todos los demás estados del flujo de trabajo constituyen un conjunto de **estados de terminación no aceptables**, en los que puede que se violen los requisitos de atomicidad de los fallos.

Los estados aceptables de terminación pueden declararse comprometidos o abortados. Un **estado aceptable de terminación comprometido** es un estado de ejecución en el que los objetivos del flujo de trabajo se han conseguido. Por el contrario, un **estado aceptable de terminación abortado** es un estado válido de terminación en el que el flujo de trabajo no ha logrado alcanzar sus objetivos. Si se ha alcanzado un estado aceptable de terminación abortado hay que deshacer todos los efectos indeseables de la ejecución parcial del flujo de trabajo de acuerdo con los requisitos de atomicidad ante fallos del flujo de trabajo.

El flujo de trabajo debe alcanzar un estado aceptable de terminación *incluso en caso de fallo del sistema*. Por tanto, si el flujo de trabajo se hallaba en un estado no aceptable de terminación en el momento del fallo, durante la recuperación del sistema hay que llevarlo a un estado aceptable de terminación (bien sea abortado, bien comprometido).

Por ejemplo, en el flujo de trabajo del procesamiento de los préstamos, en el estado final, o bien se comunica al solicitante del préstamo que no se le puede conceder, o se le abona el importe solicitado. En caso de fallo como puede ser un fallo de larga duración del sistema de verificación, puede devolverse la solicitud de préstamo al solicitante con una explicación adecuada; este resultado constituiría una terminación abortada aceptable. Una terminación comprometida aceptable sería la aceptación o el rechazo de la solicitud.

En general, las tareas pueden comprometer y liberar sus recursos antes de que el flujo de trabajo alcance un estado de terminación. Sin embargo, si la transacción multitarea aborta posteriormente, su atomicidad ante fallos puede que exija que se deshagan todos los efectos de las tareas ya completadas (por ejemplo, las subtransacciones comprometidas) ejecutando tareas compensadoras (como las subtransacciones). La semántica de la compensación exige que la transacción compensadora acabe completando su ejecución con éxito, quizás tras varios reenvíos.

En el flujo de trabajo del procesamiento de las notas de gastos, por poner un ejemplo, puede que se reduzca el importe del presupuesto del departamento debido a la aprobación inicial de una nota de gastos por el gestor. Si posteriormente se rechaza esa nota, debido a un fallo o por otro motivo, puede que haya que restaurar el presupuesto mediante una transacción compensadora.

24.2.3. Ejecución de los flujos de trabajo

La ejecución de las tareas puede controlarla un coordinador humano o un sistema de software denominado **sistema gestor de flujos de trabajo**. Los sistemas gestores de flujos de trabajo consisten en un planificador, los agentes para las tareas y un mecanismo para consultar el estado del sistema del flujo de trabajo. Cada agente de tarea controla la ejecución de una tarea por una entidad de procesamiento. El planificador es un programa que procesa los flujos de trabajo remitiendo diferentes tareas para su ejecución, controlando los diferentes eventos y evaluando las condiciones relativas a las dependencias entre las tareas. El planificador puede remitir una tarea para su ejecución (a un agente de tareas) o solicitar que se aborte una tarea previamente remitida. En el caso de las transacciones con varias bases de datos, las tareas son subtransacciones y las entidades de procesamiento son sistemas gestores de bases de datos locales. De acuerdo con las especificaciones del flujo de trabajo, el planificador hace que se cumplan las dependencias de planificación y es responsable de asegurar que las tareas alcancen estados aceptables de terminación.

Hay tres enfoques arquitectónicos del desarrollo de los sistemas gestores de flujos de trabajo. La **arquitectura centralizada** tiene un solo planificador que programa las tareas de todos los flujos de trabajo que se ejecutan de manera concurrente. La **arquitectura parcialmente distribuida** tiene un planificador para cada flujo de trabajo. Cuando los problemas de la ejecución concurrente pueden separarse de la función de planificación, esta opción es una elección natural. La **arquitectura completamente distribuida** no tiene planificador, pero los agentes de tareas coordinan su ejecución comunicándose entre sí para satisfacer las dependencias entre las tareas y otros requisitos de ejecución del flujo de trabajo.

Los sistemas de ejecución de flujos de trabajo siguen el enfoque totalmente distribuido que se acaba de describir y están basados en la mensajería. La mensajería puede implementarse mediante mecanismos de mensajería persistente. Algunas implementaciones utilizan el correo electrónico para la mensajería; estas implementaciones proporcionan muchas de las características de la mensajería persistente, pero generalmente no garantizan la atomicidad de la entrega de los mensajes y el compromiso de las transacciones. Cada sitio tiene un agente de tareas que ejecuta las tareas recibidas mediante los mensajes. Puede que la ejecución también implique la entrega de mensajes a personas, que tienen que llevar a cabo alguna acción. Cuando se completa una tarea en un sitio y hay que procesarla en otro sitio, el agente de tareas transmite un mensaje al sitio siguiente. El mensaje contiene toda la información relevante sobre la tarea que hay que realizar. Estos sistemas de flujos de trabajo basados en mensajes resultan especialmente útiles en las redes que se pueden desconectar durante parte del tiempo, como las redes de acceso telefónico.

El enfoque centralizado se utiliza en sistemas de flujos de trabajo en que los datos se almacenan en una base de datos central. El planificador notifica a los diferentes agentes, como pueden ser las personas o los programas informáticos, que hay que llevar a cabo una tarea y realiza un seguimiento de su finalización. Resulta más sencillo realizar un seguimiento del estado del flujo de trabajo con el enfoque centralizado que con el enfoque completamente distribuido.

El planificador debe garantizar que termine el flujo de trabajo en uno de los estados aceptables de terminación especificados. Idealmente, antes de intentar ejecutar un flujo de trabajo, el planificador debe examinarlo para comprobar si puede terminar en un estado no aceptable. Si el planificador no puede garantizar que el flujo de trabajo termine en un estado aceptable, debe rechazar esas especificaciones sin intentar ejecutar el flujo de trabajo. Por ejemplo, considérese un flujo de trabajo consistente en dos tareas representadas por las subtransacciones S_1 y S_2, con los requisitos de atomicidad ante fallos que indican que se deben comprometer las dos subtransacciones o ninguna de ellas. Si S_1 y S_2 no proporcionan estados preparados para comprometerse (para un compromiso de dos fases) y, además, no tienen transacciones compensadoras, es posible alcanzar un estado en que se comprometa una subtransacción y se aborte la otra, y no haya manera de llevar a las dos al mismo estado. Por tanto, esa especificación del flujo de trabajo es **insegura**, y debe rechazarse.

Los controles de seguridad como el que se acaba de describir pueden ser imposibles o poco prácticos de implementar en el planificador; pasa a ser, entonces, responsabilidad de la persona que diseña la especificación del flujo de trabajo asegurarse de que el flujo de trabajo sea seguro.

24.2.4. Recuperación de los flujos de trabajo

El objetivo de la **recuperación de los flujos de trabajo** es hacer que se cumpla la atomicidad ante fallos de los flujos de trabajo. Los procedimientos de recuperación deben asegurarse de que, si se produce un fallo en cualquiera de los componentes de procesamiento del flujo de trabajo (incluido el planificador), éste acabe alcanzando un estado aceptable de terminación (sea abortado o comprometido). Por ejemplo, el planificador puede continuar procesando tras el fallo y la recuperación, como si no hubiera pasado nada, lo que proporciona recuperabilidad hacia delante. En caso contrario, el planificador puede abortar todo el flujo de trabajo (es decir, alcanzar uno de los estados globales abortados). De cualquier forma, puede que haga falta comprometer algunas subtransacciones o incluso remitirlas para su ejecución (por ejemplo, las subtransacciones compensadoras).

Se da por supuesto que las entidades de procesamiento implicadas en el flujo de trabajo tienen sus propios sistemas locales de recuperación y tratan sus fallos locales. Para recuperar el contexto del entorno de ejecución las rutinas de recuperación de los fallos deben restaurar la información de estado del planificador en el momento del fallo, incluida la información sobre el estado de ejecución de cada tarea. Por tanto, la información de estado correspondiente debe registrarse en almacenamiento estable.

También hay que considerar el contenido de las colas de mensajes. Cuando un agente transfiere una tarea a otro, la transferencia debe ejecutarse exactamente una vez: si la transferencia tiene lugar dos veces, puede que se ejecute dos veces una tarea; si no se produce la transferencia, puede que se pierda la tarea. La mensajería persistente (Apartado 19.4.3) proporciona exactamente las características para asegurar una transferencia positiva y única.

24.2.5. Sistemas gestores de flujos de trabajo

Los flujos de trabajo suelen codificarse a mano como parte de los sistemas de aplicaciones. Por ejemplo, los sistemas de planificación de los recursos de las empresas (enterprise resource planning, ERP), que ayudan a coordinar las actividades en toda la empresa, tienen incorporados numerosos flujos de trabajo.

El objetivo de los sistemas gestores de flujos de trabajo es simplificar la construcción de flujos de trabajo y hacerlos más dignos de confianza, permitiéndoles que se especifiquen en un modo de nivel elevado y se ejecuten de acuerdo con la especificación. Hay gran número de sistemas comerciales de gestión de flujos de datos; algunos, como FlowMark de IBM, son sistemas gestores de flujos de trabajo de propósito general, mientras que otros son específicos de flujos de trabajo concretos, como los sistemas de procesamiento de órdenes o los sistemas de comunicación de fallos.

En el mundo actual de organizaciones interconectadas, no es suficiente gestionar los flujos de trabajo exclusivamente en el interior de una organización. Los flujos de trabajo que atraviesan las fronteras organizativas se están volviendo cada vez más frecuentes. Por ejemplo, considérese un pedido realizado por una organización y comunicado a otra organización que lo atiende. En cada organización puede que haya un flujo de trabajo asociado con el pedido, y es importante que los flujos de trabajo puedan operar entre sí con objeto de minimizar la intervención humana.

La Coalición de gestión de flujos de trabajo (Workflow Management Coalition) ha desarrollado estándares para la interoperatividad entre sistemas de flujos de trabajo. Los esfuerzos actuales de normalización utilizan XML como lenguaje subyacente para comunicar la información sobre el flujo de trabajo. Véanse las notas bibliográficas para obtener más información.

24.3. BASES DE DATOS EN MEMORIA PRINCIPAL

Para permitir una velocidad elevada de procesamiento de transacciones (centenares o millares de transacciones por segundo) hay que utilizar hardware de alto rendimiento y aprovechar el paralelismo. Estas técnicas, por sí solas, no obstante, resultan insuficientes para obtener tiempos de respuesta muy bajos, ya que las operaciones de E/S de disco siguen constituyendo un cuello de botella: se necesitan alrededor de diez milisegundos para cada operación de E/S y esta cifra no se ha reducido a una velocidad comparable con el aumento en la velocidad de los procesadores. Las operaciones de E/S suelen ser el cuello de botella de las operaciones de lectura y de los compromisos de las transacciones. La elevada latencia de los discos (alrededor de diez milisegundos de promedio) no sólo aumenta el tiempo necesario para tener acceso a un elemento de datos, sino que también limita el número de accesos por segundo.

Se puede hacer un sistema de bases de datos menos ligado a los discos aumentando el tamaño de la memoria intermedia de la base de datos. Los avances en la tecnología de la memoria principal permiten construir memorias principales de gran tamaño con un coste relativamente bajo. Hoy en día los sistemas comerciales de sesenta y cuatro bits pueden soportar memorias principales de decenas de gigabytes.

Para algunas aplicaciones, como el control en tiempo real, es necesario almacenar los datos en la memoria principal para cumplir los requisitos de rendimiento. El tamaño de memoria exigido para la mayoría de estos sistemas no resulta excepcionalmente grande, aunque hay unas cuantas aplicaciones que exigen que sean residentes en la memoria varios gigabytes de datos.

Dado que el tamaño de la memoria ha estado creciendo con una velocidad muy elevada, se puede esperar que un número creciente de aplicaciones tenga datos que quepan en la memoria principal.

Las memorias principales de gran tamaño permiten el procesamiento más rápido de las transacciones, ya que los datos están residentes en la memoria. No obstante, sigue habiendo limitaciones relacionadas con los discos:

- Hay que guardar en almacenamiento estable los registros del registro histórico antes de comprometer una transacción. El rendimiento mejorado que hace posible la memoria principal de gran tamaño puede hacer que el proceso de registro se convierta en un cuello de botella. Se puede reducir el tiempo de compromiso creando un búfer de registro estable en la memoria principal, utilizando RAM no volátil (implementada, por ejemplo, mediante memoria sustentada por baterías). La sobrecarga impuesta por el registro también puede reducirse mediante la técnica de *compromiso en grupo* estudiada más adelante en este apartado. La productividad (el número de transacciones por segundo) sigue estando limitada por la velocidad de transferencia de datos del disco de registro.
- Sigue habiendo que escribir los bloques de la memoria intermedia marcados como modificados por las transacciones comprometidas para que se reduzca la cantidad de registro histórico que hay que volver a ejecutar en el momento de la recuperación. Si la velocidad de actualización es extremadamente elevada, la velocidad de transferencia de los datos al disco puede convertirse en un cuello de botella.
- Si el sistema falla, se pierde toda la memoria principal. En la recuperación el sistema tiene la memoria intermedia de la base de datos vacía y hay que introducir desde el disco los elementos de datos cuando se tenga acceso a ellos. Por tanto, incluso una vez que esté completa la recuperación hace falta algo de tiempo antes de que se cargue completamente la base de datos en memoria principal y se pueda reanudar el procesamiento de transacciones de alta velocidad.

Por otro lado, las bases de datos en memoria principal ofrecen oportunidades para la optimización:

- Como la memoria resulta más costosa que el espacio de disco, hay que diseñar las estructuras internas de los datos de la memoria principal para reducir los requisitos de espacio. No obstante, las estructuras de datos pueden tener punteros que atraviesen varias páginas a diferencia de los de las bases de datos en disco, en las que el coste de que la operación de E/S atraviese varias páginas resultaría excesivamente elevado. Por ejemplo, las estructuras arbóreas de las bases de datos en memoria principal pueden ser relativamente profundas, a diferencia de los árboles B^+, pero deben minimizar los requisitos de espacio.
- No hace falta clavar en la memoria las páginas de la memoria intermedia antes de que se tenga acceso a los datos, ya que las páginas de la memoria intermedia no se sustituyen nunca.
- Las técnicas de procesamiento deben diseñarse para minimizar la sobrecarga de espacio, de modo que no se superen los límites de la memoria mientras se evalúa una consulta; esa situación daría lugar a que se paginara el área de intercambio y ralentizaría el procesamiento de la consulta.
- Una vez eliminado el cuello de botella de las operaciones de E/S del disco, pueden convertirse en cuellos de botella operaciones como los bloqueos y los pestillos. Hay que eliminar estos cuellos de botella mediante mejoras en la implementación de estas operaciones.
- Los algoritmos de recuperación pueden optimizarse, ya que rara vez hace falta borrar las páginas para hacer sitio a otras páginas.

TimesTen y DataBlitz son dos productos de bases de datos en memoria principal que aprovechan varias de estas optimizaciones, mientras que la base de datos de Oracle ha añadido características especiales para soportar memorias principales de tamaño muy grande. Se da información adicional sobre las bases de datos en memoria principal en las referencias de las notas bibliográficas.

El proceso de comprometer una transacción *T* exige que estos registros se escriban en almacenamiento estable:

- Todos los registros del registro histórico asociados con *T* que no se hayan remitidos al almacenamiento estable.
- El registro < T **comprometida** > del registro histórico.

Estas operaciones de salida suelen exigir la salida de bloques que sólo se hallan parcialmente llenos. Para asegurarse de que se saquen bloques casi llenos se utiliza la técnica de **compromiso en grupo**. En lugar de intentar comprometer *T* cuando se complete *T*, el sistema espera hasta que se hayan completado varias transacciones, o hasta que haya pasado un determinado periodo de tiempo desde que se completó la ejecución de una transacción. Luego compromete el grupo de transacciones que están esperando, todas juntas. Los bloques escritos en el registro histórico en almacenamiento estable contienen registros de varias transacciones. Mediante una cuidadosa selección del tamaño del grupo y del tiempo máximo de espera, el sistema puede asegurarse de que los bloques estén llenos cuando se escriben en

el almacenamiento estable sin hacer que las transacciones esperen demasiado. Esta técnica da como resultado, en promedio, menos operaciones de salida por cada transacción comprometida.

Aunque el compromiso en grupo reduce la sobrecarga impuesta por el registro histórico, da lugar a un ligero retraso en el compromiso de las transacciones que llevan a cabo actualizaciones. El retraso puede hacerse bastante pequeño (del orden de diez milisegundos), lo que resulta aceptable para muchas aplicaciones. Estos retrasos pueden eliminarse si los discos o los controladores de disco soportan los búferes de RAM no volátil para las operaciones de escritura. Las transacciones pueden comprometerse en cuanto la operación de escritura se lleva a cabo en la memoria intermedia de RAM no volátil. En este caso, no hay necesidad de compromiso en grupo.

Obsérvese que el compromiso en grupo resulta útil incluso en bases de datos con datos residentes en disco.

24.4. SISTEMAS DE TRANSACCIONES DE TIEMPO REAL

Las restricciones de integridad que se han considerado hasta ahora corresponden a los valores almacenados en la base de datos. En determinadas aplicaciones las restricciones incluyen **tiempos límite** en los que se tiene que haber completado una tarea. Entre estas aplicaciones están la gestión de factorías, el control del tráfico y la planificación. Cuando se incluyen tiempos límite, la corrección de la ejecución ya no es exclusivamente un problema de consistencia de la base de datos. Por el contrario, hay que preocuparse por el número de tiempos límite sobrepasados y por el tiempo que hace que se sobrepasaron. Los tiempos límite se caracterizan de la manera siguiente:

- **Tiempo límite estricto**. Pueden producirse problemas graves, como fallos del sistema, si no se completa una tarea antes de su tiempo límite.
- **Tiempo límite firme**. La tarea no tiene ningún valor si se completa después del tiempo límite.
- **Tiempo límite flexible**. La tarea tiene un valor decreciente si se completa tras el tiempo límite, y el valor se aproxima a cero a medida que aumenta el retraso.

Los sistemas con tiempos límite se denominan **sistemas de tiempo real**.

La gestión de transacciones en los sistemas de tiempo real debe tener en cuenta los tiempos límite. Si el protocolo de control de concurrencia determina que la transacción T_i debe esperar, puede hacer que T_i supere el tiempo límite. En esos casos, puede que resulte preferible adelantar la transacción que mantiene el bloqueo y permitir que T_i siga adelante. El adelantamiento debe utilizarse con cuidado, no obstante, ya que el tiempo perdido por la transacción adelantada (debido al retroceso y al reinicio) puede hacer que la transacción supere su tiempo límite. Por desgracia, es difícil determinar si es preferible retroceder o esperar en una situación dada.

Una de las principales dificultades para soportar las restricciones de tiempo real surge de la variabilidad en el tiempo de ejecución de las transacciones. En el caso más favorable, todos los accesos a los datos hacen referencia a datos de la memoria intermedia de la base de datos. En el peor de los casos, cada acceso hace que se escriba una página de la memoria intermedia en el disco (precedida de los registros del registro histórico necesario), seguido de la lectura desde el disco de la página que contiene los datos a los que hay que tener acceso. Como los dos o más accesos al disco necesarios en el peor de los casos tardan varios órdenes de magnitud más que las referencias a la memoria principal necesarias en el caso más favorable, el tiempo de ejecución de las transacciones puede estimarse con muy poca precisión si los datos están residentes en el disco. Por tanto, se suelen utilizar las bases de datos en memoria principal si hay que cumplir restricciones de tiempo real.

Sin embargo, aunque los datos estén residentes en la memoria principal, la variabilidad del tiempo de ejecución surge de las esperas de los bloqueos, de los abortos de las transacciones, etcétera. Los investigadores han dedicado esfuerzos considerables al control de concurrencia para las bases de datos de tiempo real. Han ampliado los protocolos de bloqueo para conceder una prioridad más elevada a las transacciones con tiempos límite más próximas. Han hallado que los protocolos de concurrencia optimistas tienen un buen comportamiento en las bases de datos de tiempo real; es decir, estos protocolos dan lugar a menos tiempos límite sobrepasados incluso que los protocolos de bloqueo ampliados. Las notas bibliográficas proporcionan referencias para la investigación en el área de las bases de datos de tiempo real.

En los sistemas de tiempo real, los tiempos límite, y no la velocidad absoluta, son el aspecto más importante. El diseño de sistemas de tiempo real implica asegurarse de que hay suficiente capacidad de procesamiento como para respetar las tiempos límite sin necesitar excesivos recursos de hardware. La consecución de este objetivo, pese a la variabilidad de los tiempos de ejecución resultante de la gestión de las transacciones, sigue constituyendo un problema sin resolver.

24.5. TRANSACCIONES DE LARGA DURACIÓN

El concepto de transacción se desarrolló inicialmente en el contexto de las aplicaciones de procesamiento de datos, en el que la mayor parte de las transacciones son de corta duración y no interactivas. Aunque las técnicas presentadas aquí y, anteriormente, en los Capítulos 15, 16 y 17 funcionan bien en esas aplicaciones, surgen problemas graves cuando se aplica este concepto a sistemas de bases de datos que implican la interacción con personas. Esas transacciones tienen las siguientes propiedades principales:

- **Larga duración.** Una vez que una persona interactúa con una transacción activa esa transacción se transforma en una **transacción de larga duración** desde la perspectiva de la computadora, ya que el tiempo de respuesta de las personas es lento en comparación con la velocidad de las computadoras. Además, en las aplicaciones de diseño, la actividad humana puede suponer horas, días o periodos incluso más prolongados. Por tanto, las transacciones pueden ser de larga duración en términos humanos, además de serlo en términos de la máquina.
- **Exposición de datos no comprometidos.** Los datos generados y mostrados a los usuarios por las transacciones de larga duración no están comprometidos, ya que la transacción puede abortarse. Por tanto, los usuarios —y, en consecuencia, las demás transacciones— pueden verse forzados a leer datos no comprometidos. Si varios usuarios están colaborando en un proyecto puede que las transacciones de los usuarios necesiten intercambiar datos antes de comprometer las transacciones.
- **Subtareas.** Cada transacción interactiva puede consistir en un conjunto de subtareas iniciadas por el usuario. Puede que el usuario desee abortar una subtarea sin hacer necesariamente que aborte toda la transacción.
- **Recuperabilidad.** Resulta inaceptable abortar una transacción interactiva de larga duración debido a un fallo del sistema. La transacción activa debe recuperarse hasta un estado que existiera poco antes del fallo para que se pierda una cantidad de trabajo humano relativamente pequeña.
- **Rendimiento.** El buen rendimiento de los sistemas interactivos de transacciones se define como tiempo de respuesta rápido. Esta definición difiere de la de los sistemas no interactivos, en los que el objetivo es una productividad (número de transacciones por segundo) elevada. Los sistemas con productividad elevada hacen un uso eficiente de los recursos del sistema. Sin embargo, en el caso de las transacciones interactivas, el recurso más costoso es el usuario. Si hay que optimizar la eficiencia y la satisfacción del usuario, el tiempo de respuesta debe ser rápido (desde el punto de vista humano). En los casos en los que una tarea tarda mucho tiempo, el tiempo de respuesta debe ser predecible (es decir, la variabilidad de los tiempos de respuesta debe ser baja), de modo que los usuarios puedan administrar bien su tiempo.

En los Apartados 24.5.1 a 24.5.5 se verá el motivo de que estas cinco propiedades sean incompatibles con las técnicas presentadas hasta ahora, y se estudiará el modo en que se pueden modificar esas técnicas para acomodar las transacciones interactivas de larga duración.

24.5.1. Ejecuciones no secuenciables

Las propiedades que se han estudiado hacen poco práctico obligar a que se cumpla el requisito empleado en los capítulos anteriores de que sólo se permitan las planificaciones secuenciables. Cada uno de los protocolos de control de concurrencia del Capítulo 16 tiene efectos negativos sobre las transacciones de larga duración:

- **Bloqueo de dos fases.** Cuando no se puede conceder un bloqueo, la transacción que lo ha solicitado se ve obligada a esperar a que se desbloquee el elemento de datos en cuestión. La duración de la espera es proporcional a la duración de la transacción que sostiene el bloqueo. Si el elemento de datos está bloqueado por una transacción de corta duración, se espera que el tiempo de espera sea breve (excepto en el caso de interbloqueos o de carga extraordinaria del sistema). Sin embargo, si el elemento de datos está bloqueado por una transacción de larga duración, la espera será prolongada. Los tiempo de espera elevados provocan tiempos de respuesta mayores y una mayor posibilidad de interbloqueos.
- **Protocolos basados en grafos.** Los protocolos basados en grafos permiten que se liberen los bloqueos antes que con los protocolos de bloqueo de dos fases, y evitan los interbloqueos. Sin embargo, imponen una ordenación de los elementos de datos. Las transacciones deben bloquear los elementos de datos de manera consistente con esta ordenación. En consecuencia, puede que una transacción tenga que bloquear más datos de los que necesita. Además, la transacción debe mantener el bloqueo hasta que no haya posibilidades de que se vuelva a necesitar. Por tanto, es probable que se produzcan esperas por bloqueos de larga duración.

- **Protocolos basados en las marcas temporales.** Los protocolos de marcas temporales nunca necesitan que las transacciones esperen. Sin embargo, exigen que las transacciones se aborten bajo ciertas circunstancias. Si se aborta una transacción de larga duración, se pierde una cantidad sustancial de trabajo. Para las transacciones no interactivas este trabajo perdido supone un problema de rendimiento. Para las transacciones interactivas el problema también es de satisfacción de los usuarios. Resulta muy poco deseable que el usuario descubra que se han deshecho varias horas de trabajo.
- **Protocolos de validación.** Al igual que los protocolos basados en las marcas temporales, los protocolos de validación hacen que se cumpla la secuencialidad mediante el aborto de transacciones.

Por tanto, parece que el cumplimiento de la secuencialidad provoca esperas de larga duración, el aborto de transacciones de larga duración o ambas cosas. Hay resultados teóricos, citados en las notas bibliográficas, que sustentan esta conclusión.

Surgen dificultades adicionales con el cumplimiento de la secuencialidad cuando se consideran los problemas de las recuperaciones. Ya se ha estudiado el problema de los retrocesos en cascada, en los que el aborto de una transacción puede conducir al aborto de otras transacciones. Este fenómeno no es deseable, especialmente para las transacciones de larga duración. Si se utiliza el bloqueo, se deben mantener bloqueos exclusivos hasta el final de la transacción, si hay que evitar el retroceso en cascada. Este mantenimiento de bloqueos exclusivos, no obstante, aumenta la duración del tiempo de espera de las transacciones.

Por tanto, parece que el cumplimiento de la atomicidad de las transacciones debe llevar a una mayor probabilidad de las esperas de larga duración o a crear la posibilidad de retrocesos en cascada.

Estas consideraciones son la base de los conceptos alternativos de corrección de las ejecuciones concurrentes y de la recuperación de transacciones que se considerarán en el resto de este apartado.

24.5.2. Control de concurrencia

El objetivo fundamental del control de concurrencia de las bases de datos es asegurarse de que la ejecución concurrente de las transacciones no da lugar a una pérdida de la consistencia de la base de datos. El concepto de secuencialidad puede utilizarse para conseguir este objetivo, ya que todas las planificaciones secuenciables conservan la consistencia de las bases de datos. No obstante, no todas las planificaciones que conservan la consistencia de las bases de datos son secuenciables. Por ejemplo, considérese nuevamente una base de datos bancaria que consista en dos cuentas, A y B, con el requisito de consistencia de que se conserve la suma $A + B$.

T_1	T_2
leer(A) $A := A - 50$ escribir(A)	
	leer(B) $B := B - 10$ escribir(B)
leer(B) $B := B + 50$ escribir(B)	
	leer(A) $A := A + 10$ escribir(A)

FIGURA 24.5. Planificación no secuenciable en cuanto a conflictos.

Aunque la planificación de la Figura 24.5 no es secuenciable para conflictos, pese a todo, conserva la suma de $A + B$. También ilustra dos aspectos importantes del concepto de corrección sin secuencialidad.

- La corrección depende de las restricciones de consistencia concretas de la base de datos.
- La corrección depende de las propiedades de las operaciones llevadas a cabo por cada transacción.

En general, no es posible llevar a cabo un análisis automático de las operaciones de bajo nivel de las transacciones y comprobar su efecto en las restricciones de consistencia de la base de datos. Sin embargo, hay técnicas más sencillas. Una de ellas es el empleo de las restricciones de consistencia de la base de datos como base de una división de la base de datos en subbases de datos en las que se puede administrar por separado la concurrencia. Otra es el intento de tratar algunas operaciones aparte de **leer** y de **escribir** como operaciones fundamentales de bajo nivel y ampliar el control de concurrencia para trabajar con ellas.

Las notas bibliográficas hacen referencia a otras técnicas para asegurar la consistencia sin exigir secuencialidad. Muchas de estas técnicas aprovechan variedades del control de concurrencia multiversión (véase el Apartado 17.6). A las aplicaciones de procesamiento de datos más antiguas que sólo necesitan una versión los protocolos multiversión les imponen una elevada sobrecarga de espacio para almacenar las versiones adicionales. Dado que muchas de las nuevas aplicaciones de bases de datos exigen el mantenimiento de las versiones de los datos, las técnicas de control de concurrencia que aprovechan varias versiones resultan prácticas.

24.5.3. Transacciones anidadas y multinivel

Las transacciones de larga duración pueden considerarse como conjuntos de subtareas relacionadas o subtransacciones. Al estructurar cada transacción como un conjunto de subtransacciones, se puede mejorar el

paralelismo, ya que puede que sea posible ejecutar en paralelo varias subtransacciones. Además, es posible trabajar con los fallos de las subtransacciones (debidos a abortos, fallos del sistema, etcétera) sin tener que hacer retroceder toda la transacción de larga duración.

Una transacción anidada o multinivel T consiste en un conjunto $T = \{t_1, t_2, \ldots, t_n\}$ de subtransacciones y en un orden parcial P sobre T. Cada subtransacción t_i de T puede abortar sin obligar a que T aborte. En lugar de eso, puede que T reinicie t_i o simplemente escoja no ejecutar t_i. Si se compromete t_i, esa acción no hace que t_i sea permanente (a diferencia de la situación del Capítulo 17). En vez de eso, t_i *se compromete con* T, y puede que todavía aborte (o exija compensación; véase el Apartado 24.5.4) si T aborta. La ejecución de T no debe violar el orden parcial P. Es decir, si un trazo $t_i \rightarrow t_j$ aparece en el grafo de precedencia, $t_j \rightarrow t_i$ no debe estar en el cierre transitivo de P.

El anidamiento puede tener varios niveles de profundidad, representando la subdivisión de una transacción en subtareas, subsubtareas, etcétera. En el nivel inferior de anidamiento se tienen las operaciones estándar de bases de datos **leer** y **escribir** que se han utilizado anteriormente.

Si se permite a una subtransacción de T liberar los bloqueos al completarse, T se denomina **transacción multinivel**. Cuando una transacción multinivel representa una actividad de larga duración, a veces se denomina **saga**. De manera alternativa, si los bloqueos mantenidos por la subtransacción t_i de T se asignan de manera automática a T al concluir t_i, T se denomina **transacción anidada**.

Aunque el principal valor práctico de las transacciones multinivel surja en las transacciones complejas de larga duración, se utilizará el sencillo ejemplo de la Figura 24.5 para mostrar el modo en que el anidamiento puede crear operaciones de nivel superior que pueden mejorar la concurrencia. Se reescribe la transacción T_1, mediante las subtransacciones $T_{1,1}$ y $T_{1,2}$, que llevan a cabo operaciones de suma o de resta:

- T_1 consiste en
  — $T_{1,1}$, que resta 50 de A
  — $T_{1,2}$, que suma 50 a B

De manera parecida, se reescribe la transacción T^2, mediante las subtransacciones $T^{2,1}$ y $T_{2,2}$, que también llevan a cabo operaciones de suma o de resta:

- T_2 consiste en
  — $T_{2,1}$, que resta 10 de B
  — $T_{2,2}$, que suma 10 a A

No se especifica ninguna ordenación para $T_{1,1}$, $T_{1,2}$, $T_{2,1}$ y $T_{2,2}$. Cualquier ejecución de estas subtransacciones generará un resultado correcto. La planificación de la Figura 24.5 corresponde a la planificación $<T_{1,1}, T_{2,1}, T_{1,2}, T_{2,2}>$.

24.5.4. Transacciones compensadoras

Para reducir la frecuencia de las esperas de larga duración se dispone que las actualizaciones no comprometidas se muestren a otras transacciones que se ejecuten de manera concurrente. En realidad, las transacciones multinivel pueden permitir esta exposición. No obstante, la exposición de datos no comprometidos crea la posibilidad de retrocesos en cascada. El concepto de **transacciones compensadoras** ayuda a tratar este problema.

Divídase la transacción T en varias subtransacciones $t_1, t_2, \ldots, t_n$. Una vez comprometida la subtransacción t_i, libera sus bloqueos. Ahora, si hay que abortar la transacción del nivel externo T, hay que deshacer el efecto de sus subtransacciones. Supóngase que las subtransacciones $t_1, \ldots, t_k$ se han comprometido y que t_{k+1} se estaba ejecutando cuando se tomó la decisión de abortar. Se pueden deshacer los efectos de t_{k+1} abortando esa subtransacción. Sin embargo, no es posible abortar las subtransacciones $t_1, \ldots, t_k$, puesto que ya se han comprometido.

En lugar de eso, se ejecuta una nueva subtransacción tc_i, denominada *transacción compensadora*, para deshacer el efecto de cada subtransacción t_i. Es necesario que cada subtransacción t_i tenga su transacción compensadora tc_i. Las transacciones compensadoras deben ejecutarse en el orden inverso $tc_k, \ldots, tc_1$. A continuación se ofrecen varios ejemplos de compensaciones:

- Considérese la planificación de la Figura 24.5, que se ha demostrado que es correcta, aunque no secuenciable en cuanto a conflictos. Cada subtransacción libera sus bloqueos una vez se completa. Supóngase que T_2 falla justo antes de su terminación, una vez que $T_{2,2}$ ha liberado sus bloqueos. Se ejecuta una transacción compensadora para $T_{2,2}$ que resta 10 de A y una transacción compensadora para $T_{2,1}$ que suma 10 a B.
- Considérese una inserción en la base de datos por la transacción T_i que, como efecto lateral, provoca que se actualice el índice del árbol B^+. La operación de inserción puede haber modificado varios nodos del índice del árbol B^+. Otras transacciones pueden haber leído estos nodos al tener acceso a datos diferentes del registro insertado por T_i. Como en el Apartado 17.9, se puede deshacer la inserción eliminando el registro insertado por T_i. El resultado es un árbol B^+ correcto y consistente, pero no necesariamente uno con exactamente la misma estructura que la que se tenía antes de que se iniciara T_i. Por tanto, la eliminación es una acción compensadora para la inserción.
- Considérese una transacción de larga duración T_i que represente una reserva de viaje. La transacción T tiene tres subtransacciones: $T_{i,1}$, que hace las reservas de billetes de avión; $T_{i,2}$, que reserva los coches de alquiler; y $T_{i,3}$, que reserva las habita-

ciones de hotel. Supóngase que el hotel cancela la reserva. En lugar de deshacer todas las T_i, se compensa el fallo de $T_{i,3}$ eliminando la reserva de hotel antigua y realizando una nueva.

Si el sistema falla en medio de la ejecución de una transacción del nivel externo hay que hacer retroceder sus subtransacciones cuando se recupere. Las técnicas descritas en el Apartado 17.9 pueden utilizarse con este fin.

La compensación del fallo de una transacción exige que se utilice la semántica de la transacción que ha fallado. Para determinadas operaciones, como el incremento o la inserción en un árbol B^+, la compensación correspondiente se define con facilidad. Para transacciones más complejas puede que los planificadores de la aplicación tengan que definir la forma correcta de compensación en el momento en que se codifique la transacción. Para las transacciones interactivas complejas puede que sea necesario que el sistema interactúe con el usuario para determinar la forma adecuada de compensación.

24.5.5. Problemas de implementación

Los conceptos sobre las transacciones estudiados en este apartado crean serias dificultades para su implementación. Aquí se presentan unos cuantos y se estudia el modo de abordar esos problemas.

Las transacciones de larga duración deben sobrevivir a los fallos del sistema. Se puede asegurar que lo harán llevando a cabo una operación **rehacer** con las subtransacciones comprometidas, y llevando a cabo una operación **deshacer** o una compensación para cualquier subtransacción de corta duración que estuviera activa en el momento del fallo. Sin embargo, estas acciones sólo resuelven parte del problema. En los sistemas típicos de bases de datos los datos internos del sistema como las tablas de bloqueos y las marcas temporales de las transacciones se conservan en almacenamiento volátil. Para que se pueda reanudar una transacción de larga duración tras un fallo hay que restaurar esos datos. Por tanto, es necesario registrar no sólo las modificaciones de la base de datos, sino también las modificaciones de los datos internos del sistema correspondientes a las transacciones de larga duración.

El registro histórico de las actualizaciones se hace más complicado cuando hay en la base de datos ciertos tipos de elementos de datos. Un elemento de datos puede ser un diseño CAD, el texto de un documento u otra forma de diseño compuesto. Estos elementos de datos son de gran tamaño físico. Por tanto, guardar los valores antiguos y nuevos del elemento de datos en un registro del registro histórico no resulta deseable.

Hay dos enfoques para reducir la sobrecarga de asegurar la recuperabilidad de elementos de datos de gran tamaño:

- **Registro histórico de operaciones.** Sólo se guardan en el registro histórico la operación llevada a cabo en el elemento de datos y el nombre del elemento de datos. El registro histórico de operaciones también se denomina **registro histórico lógico**. Para cada operación debe haber una operación inversa. Se lleva a cabo la operación **deshacer** utilizando la operación inversa y la operación **rehacer** utilizando la misma operación. La recuperación mediante el registro histórico de operaciones resulta más difícil, ya que **rehacer** y **deshacer** no son idempotentes. Además, el empleo del registro lógico para una operación que actualice varias páginas resulta muy complicado debido al hecho de que algunas, pero no todas, las páginas actualizadas pueden haberse escrito en el disco, por lo que resulta difícil aplicar tanto **rehacer** como **deshacer** a la operación en la imagen del disco durante la recuperación.

  El empleo del registro histórico físico de rehacer y registro histórico lógico de deshacer tal y como se describe en el Apartado 17.9 proporciona las ventajas de concurrencia del registro histórico lógico y evita los inconvenientes mencionados.
- **Registro histórico y paginación en la sombra.** El registro se utiliza para las modificaciones de elementos de datos de pequeño tamaño, pero los elementos de datos de gran tamaño se hacen recuperables mediante una técnica de paginación en la sombra (véase el Apartado 17.5). Cuando se utiliza esta técnica sólo hace falta almacenar por duplicado las páginas que se modifican realmente.

Independientemente de la técnica empleada, las complejidades introducidas por las transacciones de larga duración y los elementos de datos de gran tamaño complican el proceso de recuperación. Por tanto, resulta deseable permitir que algunos datos no esenciales queden exentos del registro, y confiar en las copias de seguridad fuera de línea y en la intervención de las personas.

24.6. GESTIÓN DE TRANSACCIONES EN VARIAS BASES DE DATOS

Hay que recordar del Apartado 19.8 que un sistema con varias bases de datos crea la ilusión de una integración lógica de las bases de datos, en un sistema de bases de datos heterogéneo en el que los sistemas locales de bases de datos pueden emplear diferentes modelos lógicos de datos y lenguajes de definición y de manipulación diferentes, y pueden diferenciarse en sus mecanismo de control de concurrencia y de gestión de las transacciones.

Los sistemas con varias bases de datos soportan dos tipos de transacciones:

1. **Transacciones locales.** Estas transacciones las ejecuta cada sistema local de base de datos fuera del control del sistema con varias bases de datos.
2. **Transacciones globales.** Estas transacciones se ejecutan bajo el control del sistema con varias bases de datos.

El sistema con varias bases de datos es consciente del hecho de que pueden ejecutarse transacciones locales en los sitios locales, pero no de las transacciones concretas que se ejecutan, ni de los datos a los que tienen acceso.

Asegurar la autonomía local de cada sistema de bases de datos exige que no se realice ningún cambio en su software. Por tanto, el sistema de bases de datos de un sitio no puede comunicarse directamente con los de otros sitios para sincronizar la ejecución de transacciones globales activas en varios sitios.

Dado que el sistema con varias bases de datos no tiene ningún control sobre la ejecución de las transacciones globales, cada sistema local debe utilizar un esquema de control de concurrencia (por ejemplo, el compromiso de dos fases o las marcas temporales) para asegurarse de que su planificación sea secuenciable. Además, en caso de bloqueo, cada sistema local debe poder protegerse contra la posibilidad de interbloqueos locales.

La garantía de la secuencialidad local no es suficiente para asegurar la secuencialidad global. Como ejemplo, considérense dos transacciones globales T_1 y T_2, cada una de las cuales tiene acceso y actualiza a dos elementos de datos, A y B, ubicados en los sitios S_1 y S_2, respectivamente. Supóngase que las planificaciones locales son secuenciables. Sigue siendo posible que haya una situación en la que, en el sitio S_1, T_2 siga a T_1, mientras que, en S_2, T_1 siga a T_2, dando como resultado una planificación global no secuenciable. En realidad, aunque no haya concurrencia entre las transacciones globales (es decir, cada transacción global sólo se remite una vez que la anterior se compromete o aborta), la secuencialidad local no resulta suficiente para asegurar la secuencialidad global (véase el Ejercicio 24.14).

En función de la implementación de los sistemas locales de bases de datos puede que una transacción global no pueda controlar el comportamiento exacto de los bloqueos de sus subtransacciones. Por tanto, incluso si todos los sistemas locales de bases de datos siguen el bloqueo de dos fases, puede que sólo sea posible asegurar que cada transacción local sigue las reglas del protocolo. Por ejemplo, puede que un sistema local de bases de datos comprometa su subtransacción y libere sus bloqueos mientras que la subtransacción de otro sistema local se sigue ejecutando. Si los sistemas locales permiten el control del comportamiento de los bloqueos y todos los sistemas siguen el bloqueo de dos fases, el sistema con varias bases de datos puede asegurar que las transacciones globales se bloqueen en la modalidad de dos fases y que los puntos de bloqueo de las transacciones en conflicto definan su orden global de secuencia. Si diferentes sistemas locales siguen diferentes mecanismos de control de concurrencia, sin embargo, esta forma directa de control global no funciona.

Hay muchos protocolos para asegurar la consistencia pese a la ejecución concurrente de las transacciones globales y locales en los sistemas con varias bases de datos. Algunos se basan en imponer las condiciones suficientes para asegurar la secuencialidad global. Otros sólo aseguran una forma de consistencia más débil que la secuencialidad, pero la consiguen por medios menos restrictivos. Se considerará uno de estos últimos esquemas: la *secuencialidad de dos niveles*. El Apartado 24.5 describe más enfoques de la consistencia sin secuencialidad; se citan otros enfoques en las notas bibliográficas.

Un problema relacionado en los sistemas de bases de datos es el del compromiso atómico global. Si todos los sistemas locales siguen el protocolo de compromiso de dos fases, puede utilizarse este protocolo para conseguir la atomicidad global. No obstante, puede que los sistemas locales no diseñados para ser parte de un sistema distribuido no puedan participar en este protocolo. Aunque un sistema local pueda soportar el compromiso de dos fases, la organización propietaria del sistema puede que no desee permitir las esperas en los casos en los que se producen bloqueos. En esos casos, pueden alcanzarse compromisos que permitan la falta de atomicidad en determinadas modalidades de fallo. En la literatura proporcionada hay un tratamiento más extenso de estos asuntos (véanse las notas bibliográficas).

24.6.1. Secuencialidad de dos niveles

La **secuencialidad de dos niveles (S2N)** asegura la secuencialidad en dos niveles del sistema:

- Cada sistema local de bases de datos asegura la secuencialidad local entre sus transacciones locales, incluidas las que forman parte de las transacciones globales.

- El sistema con varias bases de datos asegura sólo la secuencialidad entre las transacciones globales, *ignorando las ordenaciones inducidas por las transacciones locales*.

Resulta sencillo hacer que se cumpla cada uno de estos niveles de secuencialidad. Los sistemas locales ya ofrecen garantías de secuencialidad; por tanto, el primer requisito es sencillo de lograr. El segundo requisito sólo se aplica a una proyección de la planificación global en la que las transacciones locales no aparecen. Por tanto, el sistema con varias bases de datos puede asegurar el segundo requisito utilizando técnicas estándar de control de concurrencia (no importa la elección concreta de la técnica).

Los dos requisitos de S2N no resultan suficientes para asegurar la secuencialidad global. Sin embargo, con el enfoque S2N, se adopta un requisito más débil que la secuencialidad, denominado **corrección fuerte**:

1. La conservación de la consistencia tal y como especifica un conjunto de restricciones de consistencia.
2. La garantía de que el conjunto de elementos de datos leído por cada transacción es consistente.

Puede probarse que determinadas restricciones al comportamiento de las transacciones, combinadas con S2N, son suficientes para asegurar la corrección fuerte (aunque no necesariamente para asegurar la secuencialidad). Se citarán varias de estas restricciones.

En cada uno de los protocolos se distingue entre los **datos locales** y los **datos globales**. Los elementos locales de datos pertenecen a un sitio concreto y se hallan bajo el control exclusivo de ese sitio. Obsérvese que no puede haber restricciones de consistencia entre los elementos locales de datos de sitios distintos. Los elementos globales de datos pertenecen al sistema con varias bases de datos y, aunque puede que se almacenen en un sitio local, se hallan bajo el control del sistema con varias bases de datos.

El **protocolo globales-lectura** permite que las transacciones globales lean, pero no actualicen, los elementos locales de datos, mientras que impide el acceso a los datos globales por parte de las transacciones locales. El protocolo globales-lectura asegura la corrección fuerte si se cumplen todas las reglas siguientes:

1. Las transacciones locales sólo tienen acceso a los elementos locales de datos.
2. Las transacciones globales pueden tener acceso a los elementos globales de datos, y pueden leer los elementos locales de datos (aunque no deben escribirlos).
3. No hay restricciones de consistencia entre los elementos de datos locales y los globales.

El **protocolo locales-lectura** concede a las transacciones locales acceso de lectura a los datos globales, pero impide el acceso a los datos locales por las transacciones globales. En este protocolo hay que introducir el concepto de **dependencia del valor**. Cada transacción tiene una dependencia del valor si el valor que escribe en el elemento de datos de un sitio depende del valor que ha leído para el elemento de datos de otro sitio.

El protocolo locales-lectura asegura la corrección fuerte si se cumplen todas las condiciones siguientes:

1. Las transacciones locales pueden tener acceso a los elementos locales de datos y pueden leer los elementos globales de datos almacenados en ese sitio (aunque no deban escribir elementos globales de datos).
2. Las transacciones globales sólo tienen acceso a los elementos globales de datos.
3. Ninguna transacción puede tener dependencia de ningún valor.

El **protocolo globales-escritura-lectura/locales-lectura** es el más generoso en términos de acceso a los datos de los protocolos que se han considerado. Permite que las transacciones globales lean y escriban los datos locales y que las transacciones locales lean los datos globales. Sin embargo, impone tanto la condición de dependencia del valor del protocolo locales-lectura como la condición del protocolo globales-lectura de que no haya restricciones de consistencia entre los datos locales y los globales.

El protocolo globales-escritura-lectura/locales-lectura asegura la corrección fuerte si se cumplen todas las condiciones siguientes:

1. Las transacciones locales pueden tener acceso a los elementos locales de datos y pueden leer los elementos globales de datos almacenados en ese sitio (aunque no deben escribir los elementos globales de datos).
2. Las transacciones globales pueden tener acceso a los elementos globales de datos y a los elementos locales de datos (es decir, pueden leer y escribir todos los datos).
3. No hay restricciones de consistencia entre los elementos locales de datos y los globales.
4. Ninguna transacción puede tener dependencia de ningún valor.

24.6.2. Aseguramiento de la secuencialidad global

Los primeros sistemas con varias bases de datos restringían las transacciones globales a ser sólo de lectura. Así evitaban la posibilidad de que las transacciones globales introdujeran inconsistencia en los datos, pero no eran lo bastante restrictivas como para asegurar la secuencialidad global. Resulta posible realmente obtener planificaciones globales de este tipo y desarrollar un esque-

ma para asegurar la secuencialidad global, y se pide al lector que haga las dos cosas en el Ejercicio 24.15.

Hay varios esquemas generales para asegurar la secuencialidad global en entornos en los que se pueden ejecutar actualizaciones y transacciones sólo de lectura. Varios de estos esquemas se basan en la idea del **billete**. Se crea un elemento de datos especial denominado billete en cada sistema local de bases de datos. Cada transacción global que tenga acceso a los datos de un sitio debe escribir en el billete de ese sitio. Este requisito asegura que las transacciones globales entren en conflicto directamente en cada sitio que visiten. Además, el gestor de las transacciones globales puede controlar el orden en el que se secuencian las transacciones globales controlando el orden en el que tienen acceso a los billetes. Las referencias a estos esquemas aparecen en las notas bibliográficas.

Si se desea asegurar la secuencialidad global en entornos en los que no se generan en cada sitio conflictos locales directos, hay que realizar algunas suposiciones sobre las planificaciones autorizadas por el sistema local de bases de datos. Por ejemplo, si las planificaciones locales son tales que el orden de compromiso y el de secuenciación son siempre idénticos, se puede asegurar la secuencialidad controlando únicamente el orden en que se comprometen las transacciones.

El problema de los esquemas que aseguran la secuencialidad global es que pueden restringir la concurrencia de manera inadecuada. Resultan especialmente propensos a hacerlo porque la mayor parte de las transacciones remiten al sistema de bases de datos subyacente las sentencias SQL, en lugar de remitir cada uno de los pasos de **lectura**, **escritura**, **compromiso** y **aborto**. Aunque sigue siendo posible asegurar la secuencialidad global con esta suposición, el nivel de concurrencia puede ser tal que otros esquemas, como la técnica de secuencialidad de dos niveles estudiada en el Apartado 24.6.1, resulten alternativas atractivas.

24.7. RESUMEN

- Los flujos de trabajo son actividades que implican la ejecución coordinada de varias tareas llevadas a cabo por diferentes entidades de proceso. No sólo existen en las aplicaciones informáticas, sino también en casi todas las actividades de una organización. Con el auge de las redes y la existencia de numerosos sistemas autónomos de bases de datos, los flujos de trabajo ofrecen una manera adecuada de llevar a cabo las tareas que implican a varios sistemas.
- Aunque los requisitos transaccionales ACID habituales resultan demasiado estrictos o no pueden implementarse para estas aplicaciones de flujo de trabajo, los flujos de trabajo deben satisfacer un conjunto limitado de propiedades transaccionales que garantiza que no se dejen los procesos en estados inconsistentes.
- Los monitores de procesamiento de transacciones se desarrollaron inicialmente como servidores con varias hebras que podían atender a un gran número de terminales desde un único proceso. Desde entonces han evolucionado y hoy en día ofrecen la infraestructura para la creación y gestión de sistemas complejos de procesamiento de transacciones que tienen gran número de clientes y varios servidores. Proporcionan servicios como colas duraderas de las solicitudes de los clientes y de las respuestas de los servidores, el encaminamiento de los mensajes de los clientes a los servidores, la mensajería persistente, el equilibrio de carga y la coordinación del compromiso de dos fases cuando las transacciones tienen acceso a varios servidores.
- Las memorias principales de gran tamaño se aprovechan en determinados sistemas para conseguir una gran productividad del sistema. En esos sistemas el registro histórico constituye un cuello de botella. Bajo el concepto de compromiso en grupo se puede reducir el número de salidas hacia el almacenamiento estable, lo que libera este cuello de botella.
- La gestión eficiente de las transacciones interactivas de larga duración resulta más compleja debido a las esperas de larga duración y a la posibilidad de los abortos. Dado que las técnicas de control de concurrencia utilizadas en el Capítulo 16 emplean las esperas, los abortos o las dos cosas, hay que considerar técnicas alternativas. Estas técnicas deben asegurar la corrección sin exigir la secuencialidad.
- Las transacciones de larga duración se representan como transacciones atómicas con operaciones atómicas de la base de datos anidadas en el nivel inferior. Si una transacción falla, sólo se abortan las transacciones activas de corta duración. Las transacciones activas de larga duración se reanudan una vez que se han recuperado todas las transacciones de corta duración. Se necesita una transacción compensadora para deshacer las actualizaciones de las transacciones anidadas que se hayan comprometido, si falla la transacción del nivel exterior.
- En los sistemas con restricciones de tiempo real la corrección de la ejecución no sólo implica la consistencia de la base de datos, sino también el cumplimiento de las tiempos límite. La amplia variabilidad de los tiempos de ejecución de las operaciones de lectura y de escritura complica el problema de la gestión de las transacciones en sistemas con restricciones temporales.
- Los sistemas con varias bases de datos proporcionan un entorno en el que las nuevas aplicaciones de bases

de datos pueden tener acceso a los datos desde gran variedad de bases de datos existentes previamente ubicadas en varios entornos heterogéneos de hardware y de software.

- Los sistemas locales de bases de datos pueden emplear diferentes modelos lógicos y lenguajes diferentes de definición y de manipulación de datos, y puede que se diferencien en sus mecanismos de control de concurrencia y de gestión de las transacciones. Los sistemas con varias bases de datos crean la ilusión de la integración lógica de las bases de datos sin exigir su integración física.

TÉRMINOS DE REPASO

- Arquitecturas de los monitores TP
  - Proceso por cliente
  - Servidor único
  - Varios servidores, un encaminador
  - Varios servidores, varios encaminadores
- Arquitecturas de los sistemas gestores del flujo de trabajo
  - Centralizadas
  - Completamente distribuidas
  - Parcialmente distribuidas
- Aseguramiento de la secuencialidad global
- Atomicidad ante fallos del flujo de trabajo
- Autonomía
- Bases de datos en memoria principal
- Bases de datos de tiempo real
- Billete
- Cambio de contexto
- Compromiso en grupo
- Coordinación de aplicaciones
  - Gestor de recursos
  - Llamada a procedimiento remoto (Remote Procedure Call, RPC)
- Corrección fuerte
- Datos globales
- Datos locales
- Ejecuciones no secuenciables
- Estado del flujo de trabajo
  - Estados de ejecución
  - Valores de salida
  - Variables externas
- Estados de terminación del flujo de trabajo
  - Abortado
  - Aceptable
  - Comprometido
  - No aceptable
- Exposición de datos no comprometidos
- Flujos de trabajo transaccionales
  - Ejecución del flujo de trabajo
  - Entidad de procesamiento
  - Especificación del flujo de trabajo
  - Tarea
- Gestor de la cola
- Monitor TP
- Multitarea
- Protocolos
  - De dependencia del valor
  - Globales-lectura
  - Globales-escritura-lectura/locales-lectura
  - Locales-lectura
- Recuperación del flujo de trabajo
- Registro histórico lógico
- Saga
- Secuencialidad de dos niveles (S2N)
- Servidor con varias hebras
- Sistema gestor de flujos de trabajo
- Sistemas de tiempo real
- Subtareas
- Sistemas con varias bases de datos
- Tiempos límite
  - Tiempo límite estricto
  - Tiempo límite firme
  - Tiempo límite flexible
- Transacciones anidadas
- Transacciones compensadoras
- Transacciones globales
- Transacciones de larga duración
- Transacciones locales
- Transacciones multinivel

EJERCICIOS

24.1. Explíquese el modo en que los monitores TP administran los recursos de la memoria y del procesador de manera más efectiva que los sistemas operativos habituales.

24.2. Compárense las características de los monitores TP con las proporcionadas por los servidores web que soportan servlets (estos servidores se han denominado *TP-lite*).

24.3. Considérese el proceso de admisión de nuevos alumnos en la universidad (o de nuevos empleados en la organización).

a. Dese una imagen de alto nivel del flujo de trabajo comenzando por el procedimiento de matrícula de los estudiantes.
b. Indíquense los estados de terminación aceptables y los pasos que implican intervención de personas.
c. Indíquense los posibles errores (incluido el vencimiento del tiempo límite) y el modo en que se tratan.
d. Estúdiese la cantidad de flujo de trabajo que se ha automatizado en la universidad.

24.4. Al igual que los sistemas de bases de datos, los sistemas de flujo de trabajo también necesitan la gestión de la concurrencia y de la recuperación. Indíquense tres motivos por los que no se puede aplicar simplemente un sistema relacional de bases de datos empleando bloqueo de dos fases, registro histórico de operaciones físicas de deshacer y compromiso de dos fases.

24.5. Si toda la base de datos cabe en la memoria principal, indíquese si sigue haciendo falta un sistema de bases de datos para administrar los datos. Explíquese la respuesta.

24.6. Considérese un sistema de bases de datos en memoria principal que se recupera de un fallo del sistema. Explíquense las ventajas relativas de

- Volver a cargar toda la base de datos en memoria principal antes de reanudar el procesamiento de las transacciones.
- Cargar los datos a medida que los soliciten las transacciones.

24.7. En la técnica de compromiso en grupo indicar el número de transacciones que deben formar parte de un grupo. Explíquese la respuesta.

24.8. Indíquese si un sistema de transacciones de alto rendimiento es necesariamente un sistema de tiempo real. Explíquese el motivo.

24.9. En un sistema de bases de datos que utilice el registro histórico de escritura adelantada indíquese el número de accesos a disco necesarios para leer un elemento de datos en el peor caso posible. Explíquese el motivo por el que esto supone un problema para los diseñadores de sistemas de bases de datos de tiempo real.

24.10. Explíquese el motivo por el que puede que no resulte práctico exigir la secuencialidad para las transacciones de larga duración.

24.11. Considérese un proceso con varias hebras que entrega mensajes desde una cola duradera de mensajes persistentes. Pueden ejecutarse de manera concurrente diferentes hebras, que intentan entregar mensajes diferentes. En caso de fallo en la entrega, el mensaje debe restaurarse en la cola. Modélense las acciones que lleva a cabo cada hebra como una transacción multinivel, de manera que no haga falta mantener los bloqueos en la cola hasta que se entregue cada mensaje.

24.12. Discútanse las modificaciones que hay que hacer en cada uno de los esquemas de recuperación tratados en el Capítulo 17 si se permiten las transacciones anidadas. Explíquense también las diferencias que se producen si se permiten las transacciones multinivel.

24.13. Indíquese la finalidad de las transacciones compensadoras. Preséntense dos ejemplos de su empleo.

24.14. Considérese un sistema con varias bases de datos en el que se garantice que, como máximo, está activa una transacción global en un momento dado y que cada sistema local asegura la secuencialidad local.

a. Sugiéranse maneras de que el sistema con varias bases de datos pueda asegurar que haya como máximo una transacción global activa en cualquier momento dado.
b. Demuéstrese mediante un ejemplo que resulta posible que se produzca una planificación global no secuenciable pese a estas suposiciones.

24.15. Considérese un sistema con varias bases de datos en el que cada sitio local asegura la secuencialidad local y todas las transacciones globales son sólo de lectura.

a. Demuéstrese mediante un ejemplo que pueden producirse ejecuciones no secuenciables en este sistema.
b. Muéstrese la manera en que se podría utilizar un esquema de billete para asegurar la secuencialidad global.

NOTAS BIBLIOGRÁFICAS

Gray y Edwards [1995] proporcionan una introducción a las arquitecturas de los monitores TP; Gray y Reuter [1993] ofrecen una descripción detallada (y excelente) con nivel de libro de texto de los sistemas de procesamiento de transacciones, incluidos capítulos sobre los monitores TP. La descripción que se ha dado de los monitores TP se basa en estas dos fuentes. X/Open [1991] define la interfaz X/Open XA. El procesamiento de las transacciones en Tuxedo se describe en Huffman [1993]. Wipfler [1987] es uno de los textos sobre el desarrollo de aplicaciones mediante CICS.

Fischer [2001] es un manual sobre los sistemas de flujo de trabajo. Un modelo de referencia para los flujos de trabajo, propuesto por la Coalición para la gestión de flujos de trabajo (Workflow Management Coalition), se presenta en Hollinsworth [1994]. El sitio web de la coalición es www.wfmc.org. La descripción de flujos de trabajo que se ha dado sigue el modelo de Rusinkiewicz y Sheth [1995].

Reuter [1989] presenta ConTracts, un método para agrupar las transacciones en actividades con varias transacciones. Algunos problemas relativos a los flujos de trabajo se abordan en el trabajo sobre las actividades de larga duración descritas por Dayal et al. [1990] y Dayal et al. [1991]. Los autores proponen las reglas de evento-condición-acción como una técnica para especificar los flujos de trabajo. Jin et al. [1993] describen los problemas de los flujos de trabajo en las aplicaciones de telecomunicaciones.

García-Molina y Salem [1992] ofrecen una introducción a las bases de datos en memoria principal. Jagadish et al. [1993] describen un algoritmo de recuperación diseñado para las bases de datos en memoria principal. En Jagadish et al. [1994] se describe un gestor de almacenamiento para las bases de datos en memoria principal.

El procesamiento de transacciones en las bases de datos de tiempo real se estudia en Abbott y García-Molina [1999] y en Dayal et al. [1990]. Barclay et al. [1982] describen un sistema de bases de datos de tiempo real empleado en un sistema de conmutación de telecomunicaciones. Los problemas de la complejidad y de la corrección en las bases de datos de tiempo real se abordan en Korth et al. [1990b] y en Soparkar et al. [1995]. El control de concurrencia y las planificaciones en las bases de datos de tiempo real se estudian en Haritsa et al. [1990], en Hong et al. [1993] y en Pang et al. [1995]. Ozsoyoglu y Snodgrass [1995] es una reseña de la investigación en las bases de datos de tiempo real y en las bases de datos temporales.

Las transacciones anidadas y las transacciones multinivel se presentan en Lynch [1983], Moss [1982], Moss [1985], Lynch y Merritt [1986], Fekete et al. [1990b], Fekete et al. [1990a], Korth y Speegle [1994] y Pu et al. [1988]. Los aspectos teóricos de las transacciones multinivel se presentan en Lynch et al. [1988] y en Weihl y Liskov [1990]. Se han definido varios modelos de transacciones ampliadas, incluidos Sagas (García-Molina y Salem [1987]), ACTA (Chrysanthis y Ramamritham [1994]), el modelo Con-Tract (Wachter y Reuter [1992]), ARIES (Mohan et al. [1992] y Rothermel y Mohan [1989]) y el modelo NT/PV (Korth y Speegle [1994]).

El fraccionamiento de las transacciones para conseguir un rendimiento mayor se aborda en Shasha et al. [1995]. Beeri et al. [1989] presentan un modelo para la concurrencia en los sistemas de transacciones anidadas. El relajamiento de la secuencialidad se estudia en García-Molina [1983] y en Sha et al. [1988]. La recuperación en los sistemas de transacciones anidadas se estudia en Moss [1987], Haerder y Rothermel [1987] y Rothermel y Mohan [1989]. La gestión de las transacciones multinivel se estudia en Weikum [1991].

Gray [1981], Skarra y Zdonik [1989], Korth y Speegle [1988] y Korth y Speegle [1990] estudian las transacciones de larga duración. El procesamiento de las transacciones para las transacciones de larga duración se considera en Weikum y Schek [1984], Haerder y Rothermel [1987], Weikum et al. [1990] y Korth et al. [1990a]. Salem et al. [1994] presentan una extensión del bloqueo de dos fases para las transacciones de larga duración al permitir la liberación precoz de los bloqueos en ciertas circunstancias. El procesamiento de las transacciones en las aplicaciones de diseño y de ingeniería del software se estudia en Korth et al. [1988], Kaiser [1990] y Weikum [1991].

El procesamiento de las transacciones en los sistemas con varias bases de datos se estudia en Breitbart et al. [1990], Breitbart et al. [1991], Breitbart et al. [1992], Soparkar et al. [1991], Mehrotra et al. [1992b] y Mehrotra et al. [1992a]. El esquema del billete se presenta en Georgakopoulos et al. [1994]. S2N se introduce en Mehrotra et al. [1991]. Un enfoque anterior, denominado *cuasi-secuencialidad*, se presenta en Du y Elmagarmid [1989].

CAPÍTULO

25

ORACLE

Hakan Jakobsson
Oracle Corporation

Cuando se fundó Oracle en 1977 como Software Development Laboratories por Larry Ellison, Bob Miner y Ed Oates no había productos de bases de datos relacionales comerciales. La compañía, cuyo nombre cambió posteriormente a Oracle, se estableció para construir un sistema de gestión de bases de datos como producto comercial y fue la primera en lanzarlo al mercado. Desde entonces Oracle ha mantenido una posición líder en el mercado de las bases de datos relacionales, pero con el paso de los años su producto y servicios ofrecidos han crecido más allá del servicio de este campo. Aparte de las herramientas directamente relacionadas con el desarrollo y gestión de bases de datos Oracle vende herramientas de inteligencia de negocio, incluyendo sistemas de gestión de bases de datos multidimensionales y un servidor de aplicaciones con una integración cercana al servidor de la base de datos.

Aparte de los servidores y herramientas relacionados con las bases de datos, la compañía ofrece software para la planificación empresarial de recursos y gestión de relaciones con el cliente, incluyendo áreas como finanzas, recursos humanos, manufactura, márketing, ventas y gestión de cadenas de suministro. La unidad Business OnLine de Oracle ofrece servicios en estas áreas como un proveedor de servicios de aplicación.

Este capítulo cubre un subconjunto de características, opciones y funcionalidad de los productos Oracle. Continuamente se desarrollan nuevas versiones de los productos, por lo que las descripciones de los productos están sujetas a cambios. Este conjunto de características descrito aquí está basado en la primera versión de Oracle9i. Antes de abordar a fondo cada uno de los temas se resumirá la motivación de cada uno de estos tipos de datos y algunos problemas importantes del trabajo con ellos.

25.1. HERRAMIENTAS PARA EL DISEÑO DE BASES DE DATOS Y LA CONSULTA

Oracle proporciona una serie de herramientas para el diseño, consulta, generación de informes y análisis de datos para bases de datos, incluyendo OLAP.

25.1.1. Herramientas para el diseño de bases de datos

La mayor parte de las herramientas de diseño de Oracle están incluidas en Oracle Internet Development Suite. Se trata de una familia de herramientas para los distintos aspectos de desarrollo de aplicaciones, incluyendo herramientas para el desarrollo de formularios, modelado de datos, informes y consultas. La familia de productos soporta el estándar UML (véase el Apartado 2.10) para el modelado. Proporciona modelado de clases para generar código para componentes de negocio para un entorno Java así como modelado de actividades para el modelado del flujo de control de propósito general. La familia también soporta XML para el intercambio de datos con otras herramientas UML.

La principal herramienta de diseño de bases de datos en la familia es Oracle Designer, que traduce la lógica de negocio y el flujo de datos en definiciones de esquemas y guiones procedimentales para la lógica de las aplicaciones. Soporta varias técnicas de modelado tales como diagramas E-R, ingeniería de información y análisis y diseño de objetos. Oracle Designer almacena el diseño en Oracle Repository, que sirve como un único punto de metadatos para la aplicación.

Los metadatos se pueden entonces utilizar para generar formularios e informes. Oracle Repository proporciona gestión de la configuración para objetos de bases de datos, formularios, clases Java, archivos XML y otros tipos de archivos.

La familia también contiene herramientas de desarrollo de aplicaciones para generar formularios, informes, y herramientas para distintos aspectos de desarrollo basado en Java y XML. El componente de inteligencia de negocio proporciona JavaBeans para funcionalidad analítica tal como visualización de datos, consultas y cálculos analíticos.

Oracle también posee una herramienta de desarrollo de aplicaciones para el almacén de datos. Oracle Warehouse Builder. Warehouse Builder es una herramienta para el diseño e implantación de todos los aspectos de un almacén de datos, incluyendo el diseño del esquema, asignaciones de datos y transformaciones, procesamiento de carga de datos y gestión de metadatos. Ora-

cle Warehouse Builder soporta los esquemas 3FN y en estrella y puede también importar diseños desde Oracle Designer.

25.1.2. Herramientas de consulta

Oracle proporciona herramientas de consulta, generación de informes y análisis de datos ad hoc, incluyendo OLAP.

Oracle Discoverer es una herramienta basada en Web para realizar consultas, informes, análisis y publicación Web ad hoc para usuarios finales y analistas de datos. Permite a los usuarios abstraer y concretar conjuntos de resultados de datos pivote y almacenar cálculos como informes que se pueden publicar en una serie de formatos tales como hojas de datos o HTML. Discoverer contiene asistentes que ayudan a los usuarios finales a visualizar los datos como gráficos. Oracle9i soporta un amplio conjunto de funciones analíticas tales como la agregación de clasificación y traslado en SQL. La interfaz de consulta de Discoverer puede generar SQL del que se puede aprovechar su funcionalidad y puede proporcionar a los usuarios finales una rica funcionalidad analítica. Puesto que el procesamiento tiene lugar en el sistema de gestión de la base de datos relacional, Discoverer no requiere un complejo motor de cálculo en el lado del cliente, y hay una versión de Discoverer con exploración.

Oracle Express Server es un servidor de bases de datos multidimensionales. Soporta una amplia variedad de consultas analíticas, así como previsiones, modelado y gestión del escenario. Puede utilizar un sistema de gestión de bases de datos relacionales como un dorsal para almacenamiento o utilizar su propio almacenamiento multidimensional de los datos.

Con la introducción de los servicios OLAP en Oracle9i, Oracle está evitando un motor de almacenamiento separado y trasladando la mayor parte de los cálculos a SQL. El resultado es un modelo donde todos los datos residen en el sistema de gestión de la base de datos relacional, y los cálculos que no se pueden realizar en SQL se realizan en un motor de cálculo que se ejecuta en el servidor de la base de datos. El modelo también proporciona una interfaz para la programación de aplicaciones Java OLAP. Hay muchas razones para evitar un motor de almacenamiento multidimensional separado:

- Un motor relacional puede dimensionarse a conjuntos de datos mucho mayores.
- Se puede utilizar un modelo de seguridad común para las aplicaciones analíticas y el almacén de datos.
- El modelado multidimensional se puede integrar con el modelado del almacén de datos.
- El sistema de gestión de la base de datos relacional tiene un conjunto mayor de características y funcionalidad en muchas áreas tales como alta disponibilidad, copia de seguridad y recuperación y soporte para herramientas de terceros.
- No hay necesidad de formar administradores de bases de datos para dos motores de bases de datos.

El principal reto al evitar un motor de bases de datos multidimensional separado es proporcionar el mismo rendimiento. Un sistema de gestión de bases de datos mutidimensional que materializa todo o grandes partes de un cubo de datos puede ofrecer tiempos de respuesta muy cortos para muchos cálculos. Oracle ha enfocado este problema de dos formas.

- Oracle ha agregado soporte SQL para un amplio rango de funciones analíticas, incluyendo cubos, abstracciones, conjuntos de agrupación, clasificaciones (*ranks*), agregación de traslado, funciones *led* y *lag*, cajones de histograma, regresión lineal y desviación estándar, junto con la capacidad de optimizar la ejecución de dichas funciones en el motor de la base de datos.
- Oracle ha extendido las vistas materializadas para permitir funciones analíticas, en particular los conjuntos de agrupación. La capacidad de materializar partes o todo el cubo es primordial para el rendimiento de un sistema de gestión de bases de datos multidimensionales y las vistas materializadas proporcionan al sistema de gestión de bases de datos relacionales la capacidad de realizar lo mismo.

25.2. VARIACIONES Y EXTENSIONES DE SQL

Oracle9i soporta todas las características principales de SQL:1999, con algunas pequeñas excepciones tales como distintos tipos de datos. Además, Oracle soporta un gran número de otras constructoras del lenguaje, algunas de las cuales casan con SQL:1999, mientras que otras son específicas de Oracle en sintaxis o funcionalidad. Por ejemplo Oracle soporta las operaciones OLAP descritas en el Apartado 22.2, incluyendo clasificación, agregación de traslado, cubos y abstracción.

Algunos ejemplos de las extensiones SQL de Oracle son:

- **connect by,** que es una forma de recorrido de árboles que permite cálculos al estilo del cierre transitivo en una única instrucción SQL. Es una sintaxis específica de Oracle para una característica que Oracle tenía desde los años 80.
- **Upsert** e **inserciones en varias tablas.** La operación upsert combina una actualización y una inser-

ción y es útil para combinar datos nuevos con antiguos en aplicaciones de almacén de datos. Si una nueva fila tiene el mismo valor de clave que una fila antigua se actualiza la fila antigua (por ejemplo agregando los valores desde la nueva fila), en otro caso se inserta la nueva fila en la tabla. Las inserciones en varias tablas permiten actualizar varias tablas basándose en una única exploración de los nuevos datos.

- cláusula **with**, que se describe en el Apartado 4.8.2.

25.2.1. Características relacionales orientadas a objetos

Oracle tiene soporte extensivo para constructores relacionales orientados a objetos, incluyendo:

- **Tipos de objetos.** Se soporta un único modelo de herencia para las jerarquías de tipos.
- **Tipos de colecciones.** Oracle soporta **varrays**, que son arrays de longitud variable, y tablas anidadas.
- **Tablas de objetos.** Se utilizan para almacenar objetos mientras se proporciona una vista relacional de los atributos de los objetos.
- **Funciones de tablas.** Son funciones que producen conjuntos de filas como salida y se pueden utilizar en la cláusula **from** de una consulta. Las funciones de tablas se pueden anidar en Oracle. Si una función de tablas se utiliza para expresar algún formulario de transformación de datos, el anidamiento de varias funciones permite que se expresen varias transformaciones en una única instrucción.
- **Vistas de objetos.** Proporcionan una vista de tablas de objetos virtuales de datos almacenados en una tabla relacional normal. Permite acceder o ver los datos en un estilo orientado a objetos incluso si los datos están realmente almacenados en un formato relacional tradicional.
- **Métodos.** Se pueden escribir en PL/SQL, Java o C.
- **Funciones de agregación definidas por el usuario.** Se pueden utilizar en instrucciones SQL de la misma forma que las funciones incorporadas tales como **sum** y **count**.
- **Tipos de datos XML.** Se pueden utilizar para almacenar e indexar documentos XML.

Oracle tiene dos lenguajes procedimentales principales, PL/SQL y Java. PL/SQL fue el lenguaje original de Oracle para los procedimientos almacenados y tiene una sintaxis similar al utilizado en el lenguaje Ada. Java se soporta mediante una máquina virtual Java dentro del motor de la base de datos. Oracle proporciona un paquete para encapsular procedimientos, funciones y variables relacionadas en unidades únicas. Oracle soporta SQLJ (SQL incorporado en Java) y JDBC y proporciona una herramienta para generar las definiciones de clases Java correspondientes a tipos de la base de datos definidos por el usuario.

25.2.2. Disparadores

Oracle proporciona varios tipos de disparadores y varias opciones para el momento y forma en que se invocan (véase el Apartado 6.4 para una introducción a los disparadores en SQL). Los disparadores se pueden escribir en PL/SQL o Java o como llamadas a C. Para los disparadores que se ejecutan sobre instrucciones LMD tales como insert, update y delete, Oracle soporta disparadores de filas (**row**) y disparadores de instrucciones (**statement**). Los disparadores de filas se pueden ejecutar una vez por cada fila que se vea afectada (actualización o borrado, por ejemplo) por la operación LMD. Un disparador de instrucciones se ejecuta solamente una vez por instrucción. En cada caso, el disparador se puede definir tanto como un disparador *before* o *after* dependiendo de si se va a invocar antes o después de que se lleva a cabo la operación LMD.

Oracle permite la creación de disparadores **instead of** para las vistas que no pueden estar sujetas a operaciones LMD. Dependiendo de la definición de la vista puede no ser posible para Oracle traducir una instrucción LMD en una vista a modificaciones de las tablas base subyacentes sin ambigüedad. Por ello las operaciones LMD sobre vistas están sujetas a numerosas restricciones. Se puede crear un disparador **instead of** sobre una vista para especificar manualmente las operaciones sobre las tablas base que van a ocurrir en respuesta a la operación LMD sobre la vista. Oracle ejecuta el disparador en lugar de la operación LMD y por consiguiente proporciona un mecanismo de rodeo de las restricciones sobre las operaciones LMD sobre las vistas.

Oracle también tiene disparadores que ejecutan otros eventos, tales como el inicio o finalización de la base de datos, mensajes de error del servidor, inicio o finalización de sesión de un usuario e instrucciones LDD tales como las instrucciones **create**, **alter** o **drop**.

25.3. ALMACENAMIENTO E INDEXACIÓN

En la jerga de Oracle, una *base de datos* consiste en información almacenada en archivos y se accede a través de un *ejemplar*, que es un área de memoria compartida y un conjunto de procesos que interactúa con los datos en los archivos.

25.3.1. Espacios de tablas

Una base de datos consiste en una o más unidades de almacenamiento lógicas denominadas **espacios de tablas**. Cada espacio de tablas, a su vez, consiste en una o más estructuras físicas denominadas **archivos de datos**. Éstos pueden ser archivos gestionados por el sistema operativo o dispositivos en bruto.

Normalmente una base de datos Oracle tendrá los siguientes espacios de tablas:

- El espacio de tablas del **sistema**, que siempre se crea. Contiene las tablas diccionario de datos y almacenamiento para los disparadores y los procedimientos almacenados.
- Espacios de tablas creados para almacenar los datos de usuario. Aunque los datos de usuario se pueden almacenar en el espacio de tablas del **sistema** es frecuentemente deseable separar los datos de usuario de los datos del sistema. Normalmente la decisión sobre los otros espacios de tablas que se deben crear está basada en el rendimiento, disponibilidad, capacidad de mantenimiento y facilidad de administración. Por ejemplo, puede ser útil tener varios espacios de tablas para las operaciones de copia de seguridad parcial y recuperación.
- Los espacios de tablas temporales. Muchas operaciones de base de datos requieren la ordenación de los datos y la rutina de ordenación puede tener que almacenar éstos temporalmente en el disco si la ordenación no se puede realizar en memoria. Se asignan espacios de tablas temporales a la ordenación, para realizar las operaciones de gestión de espacio involucradas en un volcado a disco más eficiente.

Los espacios de tablas también se pueden utilizar como un medio para trasladar datos entre las bases de datos. Por ejemplo, es común trasladar los datos desde un sistema transaccional a un almacén de datos a intervalos regulares. Oracle permite trasladar todos los datos en un espacio de tablas de un sistema a otro sencillamente copiando los archivos y exportando e importando una pequeña cantidad de metadatos del diccionario de datos. Estas operaciones pueden ser mucho más rápidas que descargar los datos de una base de datos y después usar un descargador para insertarlos en la otra. Un requisito para esta característica es que ambos sistemas utilicen el mismo sistema operativo.

25.3.2. Segmentos

El espacio en un espacio de tablas se divide en unidades, denominadas **segmentos**, cada una de las cuales contiene datos para una estructura de datos específica. Hay cuatro tipos de segmentos

- **Segmentos de datos.** Cada tabla en un espacio de tablas tiene su propio segmento de datos donde se almacenan los datos de la tabla a menos que ésta se encuentre dividida; si esto ocurre, existe un segmento de datos por división (la división en Oracle se describe en el Apartado 25.3.10).
- **Segmentos de índices.** Cada índice en un espacio de tablas posee su propio segmento de índices, excepto los índices divididos, los cuales mantienen un segmento de índices por división.
- **Segmentos temporales.** Son segmentos utilizados cuando una operación de ordenación necesita escribir datos al disco o cuando éstos se insertan en una tabla temporal.
- **Segmentos de retroceso.** Se trata de segmentos que contienen información para deshacer los cambios de las transacciones de forma que se pueda deshacer una copia no terminada. También juegan un papel importante en el modelo de control de concurrencia en Oracle y para la recuperación de la base de datos, descrito en los Apartados 25.5.1 y 25.5.2.

Debajo del nivel de segmentos se asigna espacio a un nivel de granularidad, denominado *extensión*. Cada extensión consiste en un conjunto de *bloques* contiguos de la base de datos. Un bloque de la base de datos es el nivel más bajo de granularidad en el cual Oracle ejecuta E/S a disco. Un bloque de base de la base de datos no tiene que tener el mismo tamaño que un bloque de un sistema operativo, pero debería ser un múltiplo.

Oracle proporciona parámetros de almacenamiento que permiten un control detallado de cómo se asigna y gestiona el espacio, tales como:

- El tamaño de una extensión nueva que se va a asignar para proporcionar espacio a las filas que se insertan en una tabla.
- El porcentaje de utilización de espacio con el cual un bloque de la base de datos se considera lleno y con el cual no se introducirán más filas en ese bloque (dejando algo de espacio libre en un bloque se puede permitir que las filas existentes aumenten su tamaño cuando se realizan actualizaciones sin quedarnos sin espacio en el bloque).

25.3.3. Tablas

Una tabla estándar en Oracle está organizada en montículo; esto es, la ubicación de almacenamiento de una

fila en una tabla no está basada en los valores contenidos en la fila y se fija cuando la fila se inserta. Sin embargo, si la tabla se divide, el contexto de la fila afecta a la partición en la cual está almacenada. Hay varias características y variaciones. Oracle soporta las tablas anidadas; esto es, una tabla puede tener una columna cuyo tipo de datos sea otra tabla. La tabla anidada no se almacena en línea en la tabla padre sino que se almacena en una tabla separada.

Oracle soporta tablas temporales donde la duración de los datos es la de la transacción en la cual se insertan los datos o la sesión de usuario. Los datos son privados a la sesión y se eliminan automáticamente al final de su duración.

Una *agrupación* es otra forma de organización de los datos de la tabla (véase el Apartado 11.7). El concepto, en este contexto, no se debería confundir con otros significados de la palabra *agrupación*, tales como los relacionados con la arquitectura del ordenador. En una agrupación las filas de tablas diferentes se almacenan juntas en el mismo bloque según algunas columnas comunes. Por ejemplo, una tabla de departamento y una tabla de empleados se podrían agrupar de forma que cada fila en la tabla departamento se almacene junto con todas las filas de los empleados que trabajan en ese departamento. Los valores de la clave principal o clave externa se utilizan para determinar la ubicación de almacenamiento. Esta organización mejora el rendimiento cuando las dos tablas están combinadas pero sin un aumento de espacio de un esquema desnormalizado puesto que los valores en la tabla de departamento no están repetidos para cada empleado. Como compromiso, una consulta que involucra solamente la tabla departamento puede tener que involucrar un número sustancialmente más grande de bloques que si la tabla se almacenara sola.

La organización en agrupación implica que una fila pertenece a un lugar específico; por ejemplo, una nueva fila de empleado se debe insertar con las otras filas para el mismo departamento. Por consiguiente, es obligatorio un índice en la columna de agrupación. Una organización alternativa es una agrupación asociativa. Aquí, Oracle calcula la localización de una fila aplicando una función asociativa al valor para la columna de agrupación. La función asociativa asigna la fila a un bloque específico en la agrupación asociativa. Puesto que no es necesario el recorrido del índice para acceder a una fila según su valor de columna de agrupación, esta organización puede ahorrar cantidades significativas de E/S a disco. Sin embargo, el número de cajones asociativos y otros parámetros de almacenamiento se deben establecer cuidadosamente para evitar problemas de rendimiento debido a demasiadas colisiones o malgasto de espacio debido a cajones asociativos vacíos.

La organización según agrupación asociativa y según agrupación normal se puede aplicar a una única tabla. El almacenamiento de una tabla como una agrupación asociativa con la columna de la clave principal como la clave de la agrupación puede permitir un acceso basado en un valor de clave principal con una única E/S a disco siempre que no haya desbordamiento para ese bloque de datos.

25.3.4. Tablas organizadas con índices

En una tabla *organizada con índices* los registros se almacenan en un índice de árbol B en lugar de en un montículo. Una tabla organizada con índices requiere que se identifique una clave única para su uso como la clave del índice. Aunque una entrada en un índice normal contiene el valor de la clave y el identificador de fila de la fila indexada, una tabla organizada con índices reemplaza el identificador de fila con los valores de la columna para el resto de columnas en la tabla. Comparado con el almacenamiento de los datos en una tabla en montículo normal y la creación de un índice según las columnas clave, una tabla organizada con índices puede mejorar el rendimiento y el espacio. Consideremos la lectura de todos los valores de columna de una fila, dado su valor de clave principal. Para una tabla en montículo se requeriría un examen del índice seguido por un acceso a tabla mediante identificador de fila. Para una tabla organizada con índices solamente es necesario el examen del índice.

Los índices secundarios sobre columnas que no sean clave de una tabla organizada con índices son distintos de los índices en una tabla en montículo normal. En una tabla en montículo cada fila posee un identificador de fila fijo que no cambia. Sin embargo, un árbol B se reorganiza al crecer o disminuir cuando se insertan o borran las entradas, y no hay garantía de que una fila permanezca en una ubicación dentro de una tabla organizada con índices. Por ello, un índice secundario en una tabla organizada con índices no contiene identificadores de fila normales, sino **identificadores lógicos de fila**. Un identificador lógico de fila se compone de dos partes: un identificador de fila física correspondiente a donde la fila estaba cuando se creó el índice o la última reconstrucción y un valor para la clave única. El identificador de fila física se conoce como una «suposición», puesto que sería incorrecto si la fila se ha trasladado. En este caso, la otra parte del identificador lógico de fila, el valor de la clave para la fila, se utiliza para acceder a la misma; sin embargo, este acceso es más lento que si la suposición hubiera sido correcta, puesto que involucra un recorrido del árbol B para la tabla organizada con índices desde la raíz hasta los nodos hoja, incurriendo potencialmente en varias operaciones E/S de disco. Sin embargo, si una tabla es altamente volátil y es probable que un buen porcentaje de suposiciones sean incorrectas, puede ser mejor crear un índice secundario con solamente valores clave, puesto que el uso de una suposición incorrecta puede producir una E/S a disco malgastada.

25.3.5. Índices

Oracle soporta varios tipos distintos de índices. El tipo más comúnmente utilizado es un índice de árbol B, creado en una o varias columnas. (Nota: En la termi-

nología de Oracle, como también en varios otros sistemas de bases de datos, un índice de árbol B es lo que se denomina un índice de árbol B+ en el Capítulo 12.) Las entradas de los índices tienen el siguiente formato: para un índice en las columnas col_1, col_2 y col_3, cada fila en la tabla en donde al menos una columna tenga un valor no nulo resultaría en la entrada de índice

$$<col_1> < col_2> < col_3> < \mathit{id\text{-}fila} >$$

donde $< col_i >$ denota el valor para la columna i e $< \mathit{id\text{-}fila} >$ es la identificador de fila para la fila. Oracle puede opcionalmente comprimir el prefijo de la entrada para ahorrar espacio. Por ejemplo, si hay muchas combinaciones repetidas de valores $< col_1 > < col_2 >$, la representación de cada prefijo $< col_1 > < col_2 >$ distinto se puede compartir entre las entradas que tienen esa combinación de valores, en lugar de almacenarlo explícitamente para cada entrada. La compresión de prefijos puede llevar a ahorros de espacio sustanciales.

25.3.6. Índices de mapas de bits

Los índices de mapas de bits (descritos en el Apartado 12.9.4) utilizan una representación de mapa de bits para entradas de índice que pueden llevar a un ahorro sustancial de espacio (y, por consiguiente, ahorro de E/S a disco), cuando la columna indexada tiene un número moderado de valores distintos. Los índices de mapas de bits en Oracle utilizan la misma clase de estructura de árbol B para almacenar las entradas que un índice normal. Sin embargo, donde un índice normal en una columna tuviera entradas de la forma $< col_1 > < \mathit{id\text{-}fila} >$, una entrada de índice de mapa de bits tendría la forma

$$< col_1 > < \mathit{id\text{-}filainicial} > < \mathit{id\text{-}filafinal} >$$
$$< \mathit{mapabitscomprimido} >.$$

El mapa de bits conceptualmente representa el espacio de todas las filas posibles en la tabla entre los identificadores de la fila inicial y final. El número de tales filas posibles en un bloque depende de cuántas de ellas se pueden alojar en un bloque, lo cual va en función del número de columnas en la tabla y sus tipos de datos. Cada bit en el mapa de bits representa una fila posible en un bloque. Si el valor de la columna de esa fila es el de la entrada de índice, el bit se establece a 1. Si la fila tiene algún otro valor o la fila no existe realmente en la tabla, el bit se establece a 0 (es posible que la fila no exista realmente porque un bloque de la tabla pueda tener un número más pequeño de filas que el número que se calculó como el máximo posible). Si la diferencia es grande, el resultado pueden ser grandes cadenas de ceros consecutivos en el mapa de bits, pero el algoritmo de compresión trata dichas cadenas de ceros, por lo que el efecto negativo se limita.

El algoritmo de compresión es una variación de una técnica de compresión denominada compresión de mapas de bits alineados (Byte-Aligned Bitmap Compression, BBC). Esencialmente, una sección del mapa de bits donde la distancia entre dos unos consecutivos es suficientemente pequeña se almacena como mapas de bits. Si la distancia entre dos unos es suficientemente grande (esto es, hay un número suficiente de ceros entre ellos) se almacena el número de ceros.

Los índices de mapas de bits permiten varios índices en la misma tabla para combinarse en la misma ruta de acceso si hay varias condiciones sobre las columnas indexadas en la cláusula **where** de una consulta. Por ejemplo, para la condición

$$(col_1 = 1 \textbf{ or } col_1 = 2) \textbf{ and } col_2 > 5 \textbf{ and } col_3 <> 10$$

Oracle podría calcular las filas que coinciden con la condición ejecutando operaciones booleanas sobre los mapas de bits a partir los mapas de bits de índices sobre las tres columnas. En este caso, estas operaciones se realizarían para cada índice:

- Para el índice en col_1, se realizaría la disyunción de los valores de clave 1 y 2.
- Para el índice en col_2, todos los mapas de bits para los valores de la clave > 5 se mezclarían en una operación que corresponde a una disyunción.
- Para el índice en col_3, se obtendrían los mapas de bits para los valores 10 y **null**. Entonces, se aplicaría una conjunción sobre los resultados de los dos primeros índices, seguido por dos operaciones menos booleanas de los mapas de bits para los valores 10 y **null** para col_3.

Todas las operaciones se realizan directamente sobre la representación comprimida de los mapas de bits (no es necesaria la descompresión) y el mapa de bits resultante (comprimido) representa las filas que cumplen todas las condiciones lógicas.

La capacidad de utilizar las operaciones booleanas para combinar varios índices no está limitada a los índices de mapas de bits. Oracle puede convertir identificadores de filas a la representación de mapa de bits comprimidos, por lo que se puede utilizar un índice de árbol B normal en cualquier lugar de un árbol binario u operación de mapa de bits simplemente poniendo un operador id-fila-a-mapa-de-bits en la parte superior del acceso a índices del plan de ejecución.

Como regla nemotécnica, los índices de mapas de bits tienden a ser más eficientes en el espacio que los índices de árbol B si el número de valores distintos de la clave es menor que la mitad del número de filas en una tabla. Por ejemplo, en una tabla con un millón de filas, un índice en una columna con menos de 500.000 valores distintos probablemente sería menor si se creara como un índice de mapa de bits. Para las columnas con un número muy pequeño de valores distintos (por ejemplo, las columnas que se refieren a propiedades tales como país, estado, género, estado marital y varios

estados indicadores) un índice mapa de bits podría requerir solamente una pequeña fracción del espacio normal de un índice de árbol B normal. Cualquier ventaja en el espacio también puede dar lugar a mejoras en el rendimiento en la forma de menos operaciones E/S a disco cuando se explora el índice.

25.3.7. Índices basados en funciones

Además de crear índices sobre una o varias columnas de una tabla. Oracle permite crear índices sobre expresiones que involucran una o más columnas, tales como $col_1 + col_2 * 5$. Por ejemplo, la creación de un índice sobre la expresión *upper*(*nombre*), donde *upper* es una función que devuelve la versión en mayúsculas de una cadena y *nombre* es una columna, es posible realizar búsquedas independientes de la caja (mayúsculas o minúsculas) sobre la columna *nombre*. Con el fin de buscar todas las filas con el nombre «van Gogh» de una forma eficiente se puede utilizar la condición

upper(*nombre*)='VAN GOGH'

en la cláusula **where** de la consulta. Oracle entonces casa la condición con la definición de índice y concluye que se puede utilizar el índice para recuperar todas las filas que coincidan con «van Gogh» sin considerar las mayúsculas y minúsculas del nombre cuando se almacenó en la base de datos. Se puede crear un índice basado en función como un mapa de bits o como un índice de árbol B.

25.3.8. Índices de reunión

Un índice de reunión es un índice donde las columnas clave no están en la tabla que se referencia mediante los identificadores de filas en el índice. Oracle soporta los índices de reunión mapa de bits principalmente para su uso con esquemas en estrella (véase el Apartado 22.4.2). Por ejemplo, si hay una columna para los nombres de los productos en una tabla de la dimensión productos se podría utilizar un índice de reunión de mapas de bits sobre la tabla de hechos con esta columna clave para recuperar las filas de la tabla de hechos que corresponden a un producto con un nombre específico, aunque el nombre no esté almacenado en la tabla de hechos. La forma en la que las filas en las tablas de hechos y de la dimensión correspondientes está basada en una condición de reunión se especifica cuando se crea el índice y se convierte en parte de los los metadatos de índices. Cuando se procesa una consulta el optimizador buscará la misma condición de reunión en la cláusula where de la consulta con el fin de determinar si es aplicable el índice de reunión.

Oracle permite índices de reunión de mapa de bits para tener más de una columna clave y estas columnas pueden estar en tablas diferentes. En todos los casos las condiciones de reunión entre la tabla de hechos donde se construye el índice y las tablas dimensionales se deben referir a claves únicas en las tablas dimensionales; esto es, una fila indexada en la tabla de hechos debe corresponder a una única fila en cada una de las tablas de dimensión.

Oracle puede combinar un índice de reunión de mapa de bits en una tabla de hechos con otros índices en la misma tabla (tanto si hay índices de reunión o no) mediante el uso de operadores para las operaciones booleanas del mapa de bits. Por ejemplo, consideremos un esquema con una tabla de hechos para las ventas y tablas dimensionales para los clientes, productos y fechas. Supongamos que una consulta solicita información sobre las ventas a los clientes en un cierto código postal que compraron productos de una cierta categoría de producto durante un cierto periodo de tiempo. Si existe un índice de reunión de mapa de bits sobre varias columnas donde las columnas clave son las columnas de la tabla de dimensión restringidas (código postal, categoría de producto y fecha), Oracle puede utilizar el índice de reunión para buscar las filas en la tabla de hechos que coinciden con las condiciones de restricción. Sin embargo, si existen índices individuales sobre una única columna para las columnas clave (o un subconjunto de ellas), Oracle puede recuperar los mapas de bits de las filas de la tabla de hechos que coinciden con cada condición individual y utiliza la operación **and** booleana para generar un mapa de bits de la tabla de hechos para aquellas filas que satisfacen todas las condiciones. Si la consulta contiene condiciones sobre algunas columnas de la tabla de hechos, los índices de aquellas columnas se podrían incluir en la misma ruta de acceso, incluso si fueran índices normales de árbol B o índices de dominio (los índices de dominio se describen posteriormente en el Apartado 25.3.9).

25.3.9. Índices de dominio

Oracle permite que las tablas sean indexadas por estructuras de índices que no sean propias de Oracle. Esta característica de extensibilidad del servidor Oracle permite a los fabricantes de software desarrollar los llamados cartuchos con funcionalidad para dominios de aplicación específicos, tales como texto, datos espaciales e imágenes, con la funcionalidad de indexado más allá de la proporcionada por los tipos de índice Oracle estándar. Para implementar la lógica para crear, mantener y buscar en el índice, el diseñador de índices debe asegurar que se adhiere a un protocolo específico en su interacción con el servidor Oracle.

Un índice de dominio se debe registrar en el diccionario de datos junto con los operadores que soporta. El optimizador de Oracle considera los índices de dominio como una de las posibles rutas de acceso para una tabla. Oracle permite a las funciones de coste registrarse con los operadores de forma que el optimizador pueda comparar el coste del uso del índice de dominio con los de otras rutas de acceso.

Por ejemplo, un índice de dominio para búsquedas de texto avanzadas puede soportar un operador *contains* (contiene). Una vez que se ha registrado este operador, el índice de dominio se considerará como una ruta de acceso para una consulta como

select *
from empleados
where *contains*(*resumen*,'LINUX')

donde *resumen* es una columna de texto en la tabla empleados. El índice de dominio se puede almacenar en un archivo de datos externo o dentro de una tabla Oracle organizada con índices. Un índice de dominio se puede combinar con otros índices (mapa de bits o de árbol B) en la misma ruta de acceso con la conversión entre la representación de mapa de bits y el identificador de fila y usando operaciones booleanas del mapa de bits.

25.3.10. División en particiones

Oracle soporta varias clases de división horizontal de tablas e índices y esta característica tiene una función principal en la capacidad de Oracle de soportar bases de datos muy grandes. La capacidad de dividir una tabla o índice tiene ventajas en muchas áreas.

- La copia de seguridad y recuperación es más sencilla y rápida, puesto que se puede realizar sobre particiones individuales en lugar de sobre toda la tabla.
- Las operaciones de carga en un entorno de almacén de datos son menos intrusivas: se pueden agregar datos a una partición y entonces agregar la partición a una tabla, lo que es una operación instantánea. De igual forma, eliminar una partición con datos obsoletos desde una tabla es muy sencillo en un almacén de datos que mantenga una ventana de datos históricos.
- El rendimiento de la consulta se mejora sustancialmente, puesto que el optimizador puede reconocer que solamente se tiene que acceder a un subconjunto de las particiones de una tabla con el fin de resolver la consulta (poda de particiones). También el optimizador puede reconocer que en una reunión no es necesario intentar hacer corresponder todas las filas en una tabla con todas las filas en la otra, pero que las reuniones se necesitan realizar solamente entre pares coincidentes de divisiones (reunión por particiones).

Cada fila en una tabla dividida está asociada con una partición específica. Esta asociación está basada en la columna o columnas de la división que son parte de la definición de una tabla dividida. Hay varias formas para hacer corresponder valores de columna a divisiones, dando lugar a varios tipos de divisiones, cada una con distintas características: divisiones por rangos, asociativas, compuestas y por listas.

25.3.10.1. División por rangos

En la división por rangos los criterios de división son rangos de valores. Este tipo de división está especialmente indicado para columnas de fechas, en cuyo caso todas las filas en el mismo rango de fechas, digamos un día o un mes, pertenecen a la misma partición. En un almacén de datos donde los datos se cargan desde sistemas transaccionales a intervalos regulares, la división por rangos se puede utilizar para implementar eficientemente una ventana de datos históricos.

Cada carga de datos obtiene su nueva propia partición, haciendo que el proceso de carga sea más rápido y eficiente. El sistema realmente carga los datos en una tabla separada con la misma definición de columna que en una tabla dividida. Se puede entonces verificar la consistencia de los datos, arreglarlos e indexarlos. Después de eso el sistema puede hace de la tabla separada una nueva partición de la tabla partida mediante un sencillo cambio de los metadatos en el diccionario de datos (una operación casi instantánea).

Mientras no cambien los metadatos, el proceso de carga no afecta a los datos existentes en la tabla dividida en ningún caso. No hay necesidad de realizar ningún mantenimiento de los índices existentes como parte de la carga. Los datos antiguos se pueden eliminar de una tabla sencillamente eliminando su partición; esta operación no afecta al resto de particiones. Además, las consultas en un entorno de almacén de datos frecuentemente contienen condiciones que los restringen a un cierto periodo de tiempo, tal como una quincena o mes. Si se utiliza la división de datos por rangos el optimizador de consulta puede restringir el acceso a los datos de aquellas particiones que son relevantes a la consulta y evitar una exploración de toda la tabla.

25.3.10.2. División asociativa

En la división asociativa, una función asociativa hace corresponder filas con divisiones según los valores en las columnas de la división. Este tipo de división resulta útil principalmente cuando es importante distribuir las filas equitativamente entre las particiones o cuando las reuniones por particiones son importantes para el rendimiento de la consulta.

25.3.10.3. División compuesta

En la división compuesta la tabla se divide por rangos, pero cada partición tiene subparticiones mediante el uso de división asociativa. Este tipo de división combina las ventajas de la división por rangos y la división asociativa.

25.3.10.4. División por listas

En la división por listas los valores asociados con una partición particular están en una lista. Este tipo de división es útil si los datos en la columna de división tienen un conjunto relativamente pequeño de valores discretos. Por ejemplo, una tabla con una columna provincia

se puede partir implícitamente por región geográfica si cada lista de particiones tiene las provincias que pertenecen a la misma región.

25.3.11. Vistas materializadas

La característica de la vista materializada (véase el Apartado 3.5.1) permite almacenar el resultado de una consulta SQL y utilizarlo en un procesamiento posterior. Además, Oracle mantiene el resultado materializado, actualizándolo cuando se actualizan las tablas a las que se hicieron referencia en la consulta. Las vistas materializadas se utilizan en el almacén de datos para acelerar el procesamiento de la consulta, pero esta tecnología también se utiliza para la réplica en entornos distribuidos y móviles.

En el almacén de datos, un uso común de vistas materializadas es resumir los datos. Por ejemplo, un tipo común de consulta solicita «la suma de las ventas de cada cuatrimestre durante los últimos dos años». El precálculo de los resultados, o algún resultado parcial, de dicha consulta puede acelerar drásticamente el procesamiento de la consulta comparado a calcularlo desde cero con la agregación de todos los registros de ventas por detalle.

Oracle soporta reescrituras automáticas de las consultas que aprovechan cualquier vista materializada útil cuando se resuelve una consulta. La reescritura consiste en cambiar la consulta para utilizar la vista materializada en lugar de las tablas originales en la consulta. Además, la reescritura puede agregar reuniones adicionales o procesamiento de agregación si son necesarias para obtener el resultado correcto. Por ejemplo, si una consulta necesita las ventas por cuatrimestre, la reescritura puede aprovechar una vista que materializa las ventas por mes, añadiendo agregación adicional para abstraer los meses a cuatrimestres. Oracle tiene un tipo de objeto de metadatos denominado *dimension* que permite las relaciones jerárquicas en las tablas a definir. Por ejemplo, una tabla de la dimensión temporal en un esquema en estrella Oracle puede definir un objeto de metadatos *dimension* para especificar cómo se abstraen los días a meses, los meses a cuatrimestres, los cuatrimestres a años y así sucesivamente. De igual forma se pueden especificar las propiedades jerárquicas relacionadas con la geografía, por ejemplo, cómo los distritos de ventas se abstraen a regiones. La lógica de la reescritura de la consulta examina estas relaciones puesto que permite utilizar una vista materializada para clases más amplias de consultas.

El objeto contenedor para una vista materializada es una tabla, lo que significa que una vista materializada se puede indexar, dividir o estar sujeta a otros controles para mejorar el rendimiento de la consulta.

Cuando hay cambios en los datos de las tablas referenciadas en la consulta que define una vista materializada se debe actualizar la vista materializada para reflejar dichos cambios. Oracle soporta tanto la actualización completa de una vista materializada como una actualización rápida incremental. En una actualización completa Oracle vuelve a calcular la vista materializada desde cero, lo cual puede ser la mejor opción si las tablas subyacentes han tenido cambios significativos, por ejemplo, debidos a una carga masiva. En una actualización incremental Oracle actualiza la vista utilizando registros que fueron cambiados en las tablas subyacentes; la actualización de la vista es inmediata, esto es, se ejecuta como parte de la transacción que cambió las tablas subyacentes. La actualización incremental puede ser mejor si el número de filas que se han cambiado es pequeño. Hay algunas restricciones sobre las clases de consultas según las que una vista materializada se puede actualizar de forma incremental (y otras que indican si una vista materializada siquiera se puede crear).

Una vista materializada es similar a un índice en el sentido que, aunque puede mejorar el rendimiento de la consulta, usa espacio, y su creación y mantenimiento consume recursos. Para ayudar a resolver este compromiso Oracle proporciona un paquete que puede aconsejar al usuario de las vistas materializadas más efectivas en el coste, dada una carga de trabajo particular como entrada.

25.4. PROCESAMIENTO Y OPTIMIZACIÓN DE CONSULTAS

Oracle soporta una gran variedad de técnicas de procesamiento en su motor de procesamiento de consultas. Algunas de las más importantes se describen aquí brevemente.

25.4.1. Métodos de ejecución

Los datos se pueden acceder mediante una serie de métodos de acceso:

- **Exploración de tabla completa.** El procesador de la consulta explora toda la tabla y obtiene información sobre los bloques que forman la tabla del mapa de extensión y explorando esos bloques.
- **Exploración de índices.** El procesador crea una clave de comienzo y/o finalización a partir de las condiciones en la consulta y la utiliza para explorar una parte relevante del índice. Si hay columnas que se tienen que recuperar, que no son parte del índice, la exploración del índice sería seguida por un acceso a la tabla mediante el índice del identificador de fila. Si no hay disponible ninguna clave de inicio o parada la exploración sería una exploración de índice completa.

- **Exploración rápida completa de índices.** El procesador explora las extensiones de la misma forma que la extensión de tabla en una exploración de tabla completa. Si el índice contiene todas las columnas que se necesitan en el índice y no hay buenas claves de inicio y parada que puedan reducir significativamente esa porción del índice que se exploraría en una exploración de índices normal, este método puede ser la forma más rápida de acceder a los datos. Esto es porque la exploración rápida completa aprovecha de forma completa la E/S de disco de varios bloques. Sin embargo, a diferencia de una exploración completa normal, que recorre los bloques hoja del índice en orden, una exploración rápida completa no garantiza que la salida preserve el orden del índice.
- **Reunión de índices.** Si una consulta necesita solamente un pequeño subconjunto de columnas de una tabla ancha, pero ningún índice contiene todas estas columnas, el procesador puede utilizar una reunión de índices para generar la información relevante sin acceder a la tabla, reuniendo varios índices que contienen en conjunto las columnas necesarias. Ejecuta las reuniones como reunión por asociación sobre los identificadores de filas desde los distintos índices.
- **Acceso a agrupaciones y agrupaciones asociadas.** El procesador accede a los datos utilizando la clave de agrupación.

Oracle tiene diversas formas de combinar información desde varios índices en una única ruta de acceso. Esta posibilidad permite varias condiciones en la cláusula **where** que se pueden utilizar conjuntamente para calcular el conjunto de resultados de la forma más eficientemente posible. La funcionalidad incluye la capacidad de ejecutar las operaciones booleanas conjunción, disyunción y diferencia sobre mapas de bits que representan los identificadores de filas. Hay también operadores que hacen corresponder una lista de identificadores de filas con mapas de bits y viceversa, lo que permite que los índices de árbol B normales y los índices de mapas de bits utilicen la misma ruta de acceso. Además, para muchas consultas que involucran **count (*)** en selecciones sobre una tabla el resultado se puede calcular simplemente contando los bits activados en el mapa de bits generado mediante la aplicación de las condiciones de la cláusula **where** sin acceder a la tabla.

Oracle soporta varios tipos de reuniones en el motor de ejecución: reuniones internas, externas, semireuniones y antirreuniones (una antirreunión en Oracle devuelve las filas de la parte izquierda de la entrada que no coinciden con ninguna fila en la parte derecha de la entrada; esta operación se denomina antisemirreunión en otros libros). Evalúa cada tipo de reunión mediante uno de los tres métodos: reunión por asociación, reunión por mezcla-ordenación o reunión en bucle anidado.

25.4.2. Optimización

En el Capítulo 14 se ha discutido el tema general de la optimización de la consulta. Aquí discutimos la optimización en el contexto de Oracle.

25.4.2.1. Transformaciones de consultas

Oracle realiza la optimización de consultas en varios pasos. La mayoría de las técnicas relacionadas con las transformaciones de consultas y reescritura tienen lugar antes de la selección de la ruta de acceso, pero Oracle también soporta varios tipos de transformaciones de consultas basadas en el costo que generan un plan completo y devuelven una estimación del costo para la versión estándar de la consulta y otra que ha sufrido transformaciones avanzadas. No todas las técnicas de transformación de consultas tienen garantizado su beneficio para cada consulta, pero mediante la generación de una estimación del coste para el mejor plan sin y con la transformación aplicada, Oracle puede adoptar una decisión inteligente.

Algunos de los tipos principales de transformaciones y reescrituras soportados por Oracle son los siguientes:

- **Mezcla de vistas.** La referencia de la vista en una consulta es reemplazada por la definición de la vista. Esta transformación no es aplicable a todas las vistas.
- **Mezcla compleja de vistas.** Oracle ofrece esta característica para ciertas clases de vistas que no están sujetas a la mezcla normal de vistas puesto que tienen un **group by** o **select distinct** en la definición de la vista. Si dicha vista se combina con otras tablas, Oracle puede conmutar las reuniones y la operación de ordenación utilizada por **group by** o **distinct**.
- **Subconsultas planas.** Oracle tiene una serie de transformaciones que convierten varias clases de subconsultas en reuniones, semirreuniones o antirreuniones.
- **Reescritura de vistas materializadas.** Oracle tiene la capacidad de reescribir una consulta automáticamente para aprovechar las vistas materializadas. Si alguna parte de la consulta se puede casar con una vista materializada existente, Oracle puede remplazar esta parte de la consulta con una referencia a la tabla en la cual la vista está materializada. Si es necesario, Oracle agrega condiciones de reunión u operaciones **group by** para preservar la semántica de la consulta. Si son aplicables varias vistas materializadas, Oracle recoge la que reduce la mayor cantidad de datos que se tienen que procesar. Además, Oracle somete la consulta reescrita y la versión original al proceso completo de optimización produciendo un plan de ejecución y un coste asociado estimado para cada una. Oracle entonces decide si ejecutar la versión

original o la reescrita de la consulta según la estimación del coste.

- **Transformación en estrella.** Oracle soporta una técnica para evaluar las consultas en esquemas en estrella, conocidas como transformación en estrella. Cuando una consulta contiene una reunión de una tabla de hechos con tablas dimensionales y selecciones sobre los atributos de las tablas dimensionales, la consulta se transforma borrando la condición de la reunión entre la tabla de hechos y las tablas dimensionales y remplazando la condición de selección en cada tabla dimensional por una subconsulta del formulario:

  tabla_de_hechos.th$_i$ ***in***
  (**select** cp **from** *tabla_dimensional*$_i$
  where <condiciones sobre
  tabla_dimensional$_i$ >)

  Se genera dicha subconsulta para cada tabla dimensional que tiene algún predicado restrictivo. Si la dimensión tiene un esquema en copo de nieve (véase el Apartado 22.4), la subconsulta contendrá una reunión de las tablas aplicables que forman la dimensión.

  Oracle utiliza los valores que son devueltos desde cada subconsulta para probar un índice sobre la columna de la tabla de hechos correspondiente, obteniendo un mapa de bits como resultado. Los mapas de bits generados desde distintas subconsultas se combinan con una operación **and** de mapas de bits. El mapa de bits resultante se puede utilizar para acceder a las filas de las tablas de hechos coincidentes. Por ello, solamente se accederá a las filas en la tabla de hechos que coinciden simultáneamente en las condiciones de las dimensiones restringidas.

  Tanto la decisión de si el uso de una subconsulta para una dimensión particular es ventajoso y la decisión de si la consulta reescrita es mejor que la original están basadas en la estimación de coste del optimizador.

25.4.2.2. Selección de la ruta de acceso

Oracle tiene un optimizador basado en el costo que determina el orden de la reunión, métodos de reunión y rutas de acceso. Cada operación que el optimizador considera tiene una función de coste asociada y el optimizador intenta generar la combinación de operaciones que tiene el coste global menor.

Para estimar el coste de una operación, el optimizador considera las estadísticas que se han calculado para los objetos del esquema tales como tablas e índices. La estadística contiene información sobre el tamaño del objeto, la cardinalidad, la distribución de datos de las columnas de la tabla y cosas similares. Para la estadística de columnas, Oracle soporta histogramas equilibrados en altura e histogramas de frecuencia. Para facilitar la recogida de las estadísticas del optimizador, Oracle puede supervisar la actividad de la modificación sobre tablas y sigue la pista de aquellas tablas que han sido objeto de suficientes cambios como para que pueda ser apropiado un nuevo cálculo de las estadísticas. Oracle también sigue las columnas que se utilizan en las cláusulas **where** de las consultas, lo que hace que sean candidatas potenciales para la creación del histograma. Con una única orden un usuario puede decir a Oracle que actualice las estadísticas para aquellas tablas que han sido suficientemente cambiadas. Oracle utiliza un muestreo para acelerar el proceso de recoger la nueva estadística y elige de forma automática el menor porcentaje de la muestra que sea adecuado. También determina si la distribución de las columnas marcadas merece la creación de histogramas; si la distribución está cerca de ser uniforme Oracle utiliza una representación más sencilla de la estadística de columnas.

Oracle utiliza el coste de CPU y E/S en disco en el modelo de coste en el optimizador. Para equilibrar los dos componentes almacena las medidas sobre la velocidad de CPU y rendimiento de E/S de disco como parte de la estadística del optimizador. El paquete de Oracle para recoger la estadística del optimizador calcula estas medidas.

Para consultas que involucran un número no trivial de reuniones, el espacio de búsqueda es un tema para el optimizador de consultas. Oracle soluciona este tema de varias formas. El optimizador genera un orden inicial de la reunión y entonces decide sobre los mejores métodos de la reunión y rutas de acceso para ese orden de la reunión. Entonces cambia el orden de las tablas y determina los mejores métodos de reunión y rutas de acceso para el nuevo orden y así sucesivamente, guardando el mejor plan que se ha encontrado hasta entonces. Oracle mantiene pequeña la optimización si el número de los distintos órdenes de la reunión que se han considerado es tan grande que el tiempo gastado en el optimizador puede ser grande comparado con el que se gastaría para ejecutar el mejor plan encontrado hasta entonces. Puesto que este corte depende del coste estimado para el mejor plan encontrado hasta entonces, es importante encontrar un buen plan pronto, de forma que el optimizador se pueda parar después de un pequeño número de órdenes de la reunión, resultando un mejor tiempo de respuesta. Oracle utiliza varias heurísticas para el orden inicial para aumentar la probabilidad de que el primer orden de reunión se considere bueno.

Por cada orden de reunión que se considera, el optimizador puede hacer pasadas adicionales por las tablas para decidir los métodos de reunión y las rutas de acceso. Tales pasadas adicionales capturarían efectos globales colaterales específicos sobre la selección de la ruta de acceso. Por ejemplo, una combinación específica de métodos de reunión y rutas de acceso pueden eliminar la necesidad de ejecutar una ordenación **order by**. Puesto que tal efecto lateral global puede no ser obvio cuando se consideran localmente los costes de los distintos

métodos de reunión y de rutas de acceso, se utiliza una pasada separada que capture un efecto colateral específico para encontrar un posible plan de ejecución con un mejor coste conjunto.

25.4.2.3. Poda de particiones

Para tablas divididas el optimizador intenta ajustar las condiciones en la cláusula **where** de una consulta con el criterio de división de la tabla con el fin de evitar acceder a particiones que no son necesarias para el resultado. Por ejemplo, si una tabla se divide por el rango de fechas y la consulta se restringe a datos entre dos fechas específicas, el optimizador determina las particiones que contienen los datos entre las fechas específicas y asegura que solamente se accede a dichas particiones. Este escenario es muy común y la aceleración puede ser dramática si solamente es necesario un pequeño subconjunto de particiones.

25.4.3. Ejecución en paralelo

Oracle permite ejecutar en paralelo una única instrucción SQL mediante la división del trabajo entre varios procesos en una computadora multiprocesadora. Esta característica es especialmente útil para operaciones intensivas en cómputo que de otra forma se ejecutarían en un tiempo inaceptablemente largo. Ejemplos representativos son las consultas de apoyo para la toma de decisiones que necesitan procesar grandes cantidades de datos, cargas de datos en un almacén de datos y creación o reconstrucción de índices.

Con el fin de lograr una buena aceleración mediante el paralelismo es importante que el trabajo involucrado en la ejecución de la instrucción se divida en gránulos que se pueden procesar de forma independiente mediante los distintos procesadores en paralelo. Dependiendo del tipo de operación Oracle tiene diversas formas de dividir el trabajo.

Para operaciones que acceden a objetos base (tablas e índices) Oracle puede dividir el trabajo mediante trozos horizontales de datos. Para algunas operaciones tales como una exploración completa de una tabla, cada uno de dichos trozos puede ser un rango de bloques (cada proceso de consulta en paralelo explora la tabla desde el bloque al comienzo del rango hasta el bloque al final). Para otras operaciones en una tabla dividida, como la actualización y borrado, el trozo podría ser una partición. Para inserciones en una tabla no dividida los datos a insertar se dividen de forma aleatoria entre los procesos en paralelo.

Las reuniones se pueden realizar en paralelo de distintas formas. Una forma es dividir una de las entradas a la reunión entre procesos paralelos y permitir que cada proceso reúna su trozo con la otra entrada de la reunión; éste es el método de reunión con fragmentos y réplicas del Apartado 20.5.2. Por ejemplo, si una tabla grande se reúne con una pequeña mediante una reunión por asociación, Oracle divide la tabla grande entre los procesos y envía una copia de la tabla pequeña a cada proceso, la cual entonces reúne su trozo con la tabla menor. Si ambas tablas son grandes sería prohibitivamente costoso enviar una de ellas a todos los procesos. En ese caso Oracle logra el paralelismo mediante la división de los datos entre los procesos mediante la asociación de los valores de las columnas de la reunión (el método de reunión por asociación dividida del Apartado 20.5.2.1). Cada tabla se explora en paralelo mediante un conjunto de procesos y cada fila en la salida se pasa a un proceso de un conjunto de procesos que van a ejecutar la reunión. El proceso que obtiene la fila se determina mediante una función de asociación sobre los valores de la columna de reunión. Por ello, cada proceso de reunión obtiene solamente las filas que podrían potencialmente coincidir y filas correspondientes no podrían ir a parar a diferentes procesos.

Oracle organiza en paralelo las operaciones de ordenación mediante los rangos de valores de la columna en la cual se ejecuta la ordenación (esto es, usando la ordenación de división por rangos del Apartado 20.5.1). A cada proceso que participa en la ordenación se le envían filas con los valores en este rango y ordena las filas en su rango. Para maximizar las ventajas del paralelismo las filas se tienen que dividir lo más equitativamente posible entre los procesos en paralelo y entonces surge el problema de determinar las fronteras de rango que generan una buena distribución. Oracle soluciona el problema mediante un muestreo dinámico de un subconjunto de las filas en la entrada a la ordenación antes de decidir las fronteras del rango.

25.4.3.1. Estructura del proceso

Los procesos involucrados en la ejecución en paralelo de una instrucción SQL consisten en un proceso coordinador y una serie de procesos servidores en paralelo. El coordinador es responsable de asignar trabajos a los servidores en paralelo y de recoger y devolver los datos a los procesos del usuario que enviaron la instrucción. El grado de paralelismo es el número de procesos servidores en paralelo que se asignan para ejecutar una operación primitiva como parte de la instrucción. El grado de paralelismo se determina mediante el optimizador, pero se puede reducir dinámicamente si la carga en el sistema aumenta.

Los servidores en paralelo operan sobre un modelo productor/consumidor. Cuando es necesario una secuencia de operaciones para procesar una instrucción, el conjunto productor de servidores ejecuta la primera operación y pasa los datos resultantes al conjunto de consumidores. Por ejemplo, si una exploración de tabla completa es seguida por una ordenación y el grado de paralelismo es 12, habría 12 servidores productores que ejecutan la exploración de la tabla y pasan el resultado a 12 servidores consumidores que ejecutan la ordenación. Si es necesaria una operación posterior, como otra ordenación, las funciones de los dos conjuntos de servidores se cambian. Los servidores que originalmente

ejecutaban la exploración de la tabla adoptan la función de consumidores de la salida producida por la primera ordenación y lo utilizan para ejecutar la segunda ordenación. Por ello se realizan una secuencia de operaciones pasando los datos entre dos conjuntos de servidores que alternan sus funciones como productores y consumidores. Los servidores se comunican entre sí mediante las memorias intermedias sobre hardware de memoria compartida y mediante las conexiones de red de alta velocidad sobre configuraciones MPP (sin compartimiento) y sistemas agrupados (discos compartidos).

Para sistemas sin compartimiento el coste para acceder a los datos en el disco no es uniforme entre los procesos. Un proceso que se ejecuta en un nodo que tiene acceso directo a un dispositivo puede procesar los datos sobre ese dispositivo más rápidamente que un proceso que tiene que recuperar los datos a través de la red. Oracle utiliza el conocimiento sobre la afinidad dispositivo a nodo y dispositivo a proceso (esto es, la capacidad de acceder a los dispositivos directamente) cuando distribuye el trabajo entre servidores en ejecución paralela.

25.5. CONTROL DE CONCURRENCIA Y RECUPERACIÓN

Oracle soporta técnicas de control de concurrencia y recuperación que proporcionan una serie de características útiles.

25.5.1. Control de concurrencia

El control de concurrencia multiversión de Oracle difiere de los mecanismos de concurrencia utilizados por la mayoría de los fabricantes de bases de datos. Para las consultas de sólo lectura se proporcionan instantáneas consistentes en lectura, que son vistas de la base de datos tal como existía en un cierto momento, conteniendo todas las actualizaciones que se comprometieron hasta ese momento y no el resto. Por ello no se utilizan los bloqueos de lectura y las consultas de sólo lectura no interfieren con otra actividad de la base de datos en términos de bloqueos (esto es básicamente el protocolo de bloqueo multiversión en dos fases descrito en el Apartado 16.5.2).

Oracle soporta la consistencia de lectura en un nivel de instrucción y de transacción. Al comienzo de la ejecución de una instrucción o transacción (dependiendo del nivel de consistencia que se utilice) Oracle determina el número de cambio del sistema (System Change Number, SCN) actual. El SCN esencialmente actúa como una marca temporal donde el tiempo se mide en términos de compromisos de la base de datos en lugar del tiempo de reloj.

Si en el curso de una consulta se encuentra que un bloque de datos tiene un SCN mayor que el que está siendo asociado con la consulta, es evidente que se ha modificado el bloque de datos después del SCN de la consulta original mediante alguna otra transacción y puede o no haberse comprometido. Por ello los datos en el bloque no se pueden incluir en una vista consistente de la base de datos como existía a la hora del SCN de la consulta. En su lugar, se debe utilizar una versión anterior de los datos en el bloque; específicamente aquel que tenga el SCN mayor que no exceda el SCN de la consulta. Oracle recupera la versión de los datos desde el segmento de retroceso (los segmentos de retroceso se describen en el Apartado 25.5.2). Por ello, supuesto que el segmento de retroceso es lo suficientemente grande, Oracle puede devolver un resultado consistente de la consulta incluso si los datos se han modificado varias veces desde que comenzara la ejecución de la consulta. Si el bloque con el SCN deseado ya no existe en el segmento de retroceso, la consulta devolverá un error. Habría una indicación de que el segmento de retroceso no se ha dimensionado adecuadamente, dada la actividad del sistema.

En el modelo de concurrencia de Oracle las operaciones de lectura no bloquean las operaciones de escritura y las operaciones de escritura no bloquean las operaciones de lectura, una propiedad que permite un alto grado de concurrencia. En particular, el esquema permite consultas largas (por ejemplo consultas de informes) para ejecutar en un sistema con una gran cantidad de actividad transaccional. Esta clase de escenario es normalmente problemático para sistemas de bases de datos donde las consultas utilizan bloqueos de lectura, puesto que la consulta puede fallar al adquirirlos o bloquear grandes cantidades de datos por mucho tiempo evitando, por consiguiente, la actividad transaccional de los datos y reduciendo la concurrencia (una alternativa que se utiliza en algunos sistemas es utilizar un grado inferior de consistencia, tal como la consistencia en grado dos, pero eso podría producir resultados inconsistentes en la consulta).

El modelo de concurrencia de Oracle se utiliza como base para la característica *Flashback Query*. Esta característica permite a un usuario establecer un cierto número SCN o tiempo de reloj en su sesión y ejecutar consultas sobre los datos que existían en esa fecha (supuesto que los datos todavía existían en el segmento de retroceso). Normalmente en un sistema de bases de datos, una vez que se ha realizado el cambio no hay forma de retroceder al estado anterior de los datos a menos que se realicen restauraciones desde copias de seguridad. Sin embargo, la recuperación de una base de datos muy grande puede ser muy costosa, especialmente si el objetivo es solamente recuperar algunos datos que han sido borrados inadvertidamente por un usuario. La característica *Flashback Query* proporciona un mecanismo mucho más sencillo para tratar los errores del usuario.

Oracle soporta dos niveles de aislamiento ANSI/ISO «con compromiso de lectura» y «secuenciable». No hay soporte para lecturas no actualizadas, puesto que no existe necesidad. Los dos niveles de aislamiento corresponden a si se utiliza la consistencia de la lectura en el nivel de instrucción o en el nivel de transacción. El nivel se puede establecer para una sesión o una transacción individual. La consistencia de lectura en el nivel de la instrucción es el predeterminado.

Oracle utiliza un bloqueo en el nivel de las filas. Las actualizaciones de distintas filas no entran en conflicto. Si dos escritores intentan modificar la misma fila, uno espera hasta que el otro comprometa o retroceda y entonces puede devolver un error de conflicto de escritura o seguir y modificar la fila. Los bloqueos se mantienen mientras dure la transacción.

Además de los bloqueos en el nivel de las filas que evitan las inconsistencias debido a la actividad, el LMD de Oracle utiliza los bloqueos de tabla para evitar las inconsistencias debido a la actividad LDD. Estos bloqueos evitan que, por ejemplo, un usuario elimine una tabla mientras otro usuario tiene una transacción aún no comprometida que está accediendo a la tabla. Oracle no utiliza el dimensionamiento de bloqueos para convertir los bloqueos de filas a bloqueos de tabla con el propósito de su control de concurrencia normal.

Oracle detecta los interbloqueos automáticamente y los resuelve retrocediendo una de las transacciones involucradas en el interbloqueo.

Oracle soporta transacciones autónomas que son transacciones independientes generadas con otras transacciones. Cuando Oracle invoca a una transacción autónoma genera una nueva transacción en un contexto separado. La nueva transacción se puede comprometer o retroceder antes de que el control vuelva a la transacción llamadora. Oracle soporta varios niveles de anidamiento de transacciones autónomas.

25.5.2. Estructuras básicas de recuperación

Con el fin de comprender cómo se recupera Oracle de un fallo, tal como una caída del disco, es importante comprender las características básicas que están involucradas. Además de los archivos de datos que contienen las tablas e índices hay archivos de control, registros históricos rehacer, registros históricos rehacer archivados y segmentos de retroceso.

El archivo de control contiene varios metadatos que son necesarios para operar en la base de datos, incluyendo la información sobre las copias de seguridad.

Oracle registra cualquier modificación transaccional de una memoria intermedia de la base de datos en el registro histórico rehacer, que consiste en dos o más archivos. Registra la modificación como parte de la operación que la causa y sin considerar si la transacción finalmente se produce. Registra los cambios de los índices y segmentos de retroceso así como los cambios a la tabla de datos. Cuando se llenan los registros históricos rehacer se archivan mediante uno o varios procesos en segundo plano (si la base de datos se ejecuta en modo **archivelog**).

El segmento de retroceso contiene información sobre versiones anteriores de los datos (esto es, información para deshacer). Además de esta función en el modelo de consistencia de Oracle, la información se utiliza para restaurar la versión anterior de los datos cuando se deshace una transacción que ha modificado los datos.

Para poder recuperar un fallo de almacenamiento se debería realizar una copia de seguridad de los archivos de datos y archivos de control periódicamente. La frecuencia de la copia de seguridad determina el tiempo mayor de recuperación, puesto que lleva más tiempo la recuperación si la copia de seguridad es antigua. Oracle soporta copias de seguridad en caliente (copias de seguridad ejecutadas en una base de datos en línea que está sujeta a una actividad transaccional). Durante la recuperación de una copia de seguridad, Oracle ejecuta dos pasos para alcanzar un estado consistente de la base de datos como existía antes del fallo. En primer lugar Oracle rehace las transacciones aplicando los archivos históricos rehacer (archivados) a la copia de seguridad. Esta acción lleva a la base de datos a un estado que existía en la fecha del fallo, pero no necesariamente un estado consistente, puesto que los registros históricos deshacer incluyen datos no comprometidos. En segundo lugar, Oracle deshace las transacciones no comprometidas mediante el uso del segmento de retroceso. La base de datos está ahora en un estado consistente.

La recuperación en una base de datos que ha sido objeto de una actividad transaccional grande debido a la ultima copia de seguridad puede ser costosa en tiempo. Oracle soporta recuperación en paralelo en la cual se utilizan varios procesos para aplicar información de rehacer simultáneamente. Oracle proporciona una herramienta GUI, el gestor de recuperación, que automatiza la mayor parte de las tareas asociadas con copias de seguridad y recuperación.

25.5.3. Bases de datos en espera gestionadas

Para asegurar una alta disponibilidad, Oracle proporciona la característica bases de datos en espera gestionadas (esta característica es la misma que las copias de seguridad remotas, descrita en el Apartado 17.10). Una base de datos en espera es una copia de la base de datos normal que se instala en un sistema separado. Si ocurre un fallo catastrófico en el sistema principal el sistema en espera se activa y asume el control, minimizando el efecto del fallo en la disponibilidad. Oracle mantiene la base de datos en espera actualizada mediante la aplicación constante de archivos históricos rehacer archivados que se envían desde la base de datos principal. La base de datos de seguridad se puede usar en línea en modo sólo lectura y utilizarla para informes y consultas para el apoyo a la toma de decisiones.

25.6. ARQUITECTURA DEL SISTEMA

Siempre que una aplicación de base de datos ejecuta una instrucción SQL hay un proceso del sistema operativo que ejecuta código en el servidor de bases de datos. Oracle se puede configurar de forma que el proceso del sistema operativo esté *dedicado* exclusivamente a la instrucción que se está procesando o de forma que el proceso se pueda compartir entre varias instrucciones. La última configuración, conocida como *servidor multienhebrado*, tiene propiedades diferentes respecto a la arquitectura del proceso y memoria. Se discutirá la arquitectura del servidor dedicado en primer lugar y posteriormente la arquitectura del servidor multienhebrado.

25.6.1. Servidor dedicado: estructuras de memoria

La memoria utilizada por Oracle se divide principalmente en tres categorías: áreas de código software, área global del sistema (System Global Area, SGA) y el área global del programa (Program Global Area, PGA).

Las áreas de código del sistema son las partes de la memoria donde reside el código del servidor Oracle. Se asigna un PGA para cada proceso para albergar sus datos locales e información de control. Esta área contiene espacio en pilas para diversos datos de la sesión y la memoria privada para la instrucción SQL que se está ejecutando. También contiene memoria para operaciones de ordenación y asociación que pueden ocurrir durante la evaluación de la instrucción.

SGA es un área de memoria para estructuras que son compartidas entre los usuarios. Está formada por varias estructuras principales, incluyendo:

- **Caché de memoria intermedia:** Esta caché mantiene bloques de datos a los que se accede frecuentemente (tablas e índices) en memoria para reducir la necesidad de ejecutar E/S a disco físico. Se usa una política menos actualizada excepto para bloques accedidos durante una exploración de tabla completa. Sin embargo, Oracle permite crear varias colas de memoria intermedia que tienen distintos criterios para la datación de los datos. Algunas operaciones Oracle omiten la caché de memoria intermedia y leen los datos directamente del disco.
- **Memoria intermedia de registro histórico rehacer.** Esta memoria intermedia contiene la parte del registro histórico rehacer que no se ha escrito todavía en el disco.
- **Cola compartida.** Oracle busca maximizar el número de usuarios que pueden utilizar la base de datos concurrentemente minimizando la cantidad de memoria que es necesaria para cada usuario. Un concepto importante en este contexto es la capacidad de compartir la representación interna de instrucciones SQL y el código procedimental escrito en PL/SQL. Cuando varios usuarios ejecutan la misma instrucción SQL pueden compartir la mayoría de estructuras de datos que representan el plan de ejecución de la instrucción. Solamente los datos que son locales a cada invocación específica de la instrucción necesitan mantenerse en una memoria privada.

Las partes que se pueden compartir de las estructuras de datos que representan la instrucción SQL se almacenan en la cola compartida, incluyendo el texto de la instrucción. El almacenamiento en caché de instrucciones SQL en la cola compartida también se guarda en tiempo de compilación, puesto que una nueva invocación de la instrucción que ya está almacenada en caché no tiene que pasar por el proceso de compilación completo. La determinación de si una instrucción SQL es la misma que la existente en la cola compartida se basa en la coincidencia exacta del texto y el establecimiento de ciertos parámetros de sesión. Oracle puede remplazar automáticamente las constantes en una instrucción SQL con variables vinculadas; las consultas futuras que son iguales excepto por los valores de constantes coincidirán con la consulta anterior en la cola compartida. La cola compartida también contiene cachés para información de diccionario y diversas estructuras de control.

25.6.2. Servidor dedicado: estructuras de proceso

Hay dos tipos de procesos que ejecutan código servidor Oracle: procesos servidor que procesan instrucciones SQL y procesos en segundo plano que ejecutan diversas tareas administrativas relacionadas con el rendimiento. Algunos de estos procesos son opcionales y en algunos casos se pueden utilizar varios procesos del mismo tipo por razones del rendimiento. Algunos de los tipos más importantes de procesos en segundo plano son:

- **Escritor de la base de datos.** Cuando una memoria intermedia se elimina de la caché de la memoria intermedia se debe volver a escribir en el disco si se ha modificado desde que se introdujo en la caché.

  Los procesos del escritor de la base de datos ejecutan esta tarea, lo que ayuda al rendimiento del sistema liberando espacio en la caché de la memoria intermedia.
- **Escritor del registro histórico.** El escritor del registro histórico procesa las entradas de escritura de la memoria intermedia del registro histórico rehacer al archivo del registro histórico rehacer en

el disco. También escribe un registro de compromiso al disco siempre que se compromete una transacción.

- **Punto de revisión.** El proceso punto de revisión actualiza las cabeceras del archivo de datos cuando ocurre un punto de revisión.
- **Monitor del sistema.** Este proceso realiza la recuperación ante una caída en caso necesario. También ejecuta cierta administración del espacio para reclamar espacio no utilizado en espacios temporales.
- **Monitor de procesos.** Este proceso ejecuta recuperación de procesos para procesos del servidor que fallan, liberando recursos y ejecutando diversas operaciones de limpieza.
- **Recuperador.** El proceso recuperador resuelve los fallos y dirige la limpieza de transacciones distribuidas.
- **Archivador.** El archivador copia el archivo de registro histórico rehacer en línea a un registro histórico rehacer cada vez que se llena el archivo de registro histórico en línea.

25.6.3. Servidor multienhebrado

La configuración de servidor multienhebrado aumenta el número de usuarios que un número dado de procesos servidor pueden soportar compartiendo los procesos servidor entre las instrucciones. Difiere de la arquitectura de servidor dedicado en los siguientes aspectos principales:

- Un proceso de envío en segundo plano encamina las solicitudes de usuarios al siguiente proceso servidor disponible. Al realizar esto utiliza una cola de solicitudes y una cola de respuestas en el SGA. El distribuidor pone una nueva solicitud en la cola de solicitudes donde será recogida por un proceso servidor. Un proceso servidor completa una solicitud, pone el resultado en la cola de respuestas para ser recogida por el distribuidor y ser devuelta al usuario.
- Puesto que un proceso servidor se comparte entre varias instrucciones SQL, Oracle no mantiene datos privados en el PGA. Almacena los datos específicos de la sesión en el SGA.

25.6.4. Agrupaciones de aplicaciones reales de Oracle9i

Las agrupaciones de aplicaciones reales de Oracle9i (Oracle9i Real Application Clusters) es una característica que permite que varios ejemplares de Oracle se ejecuten en la misma base de datos (recuérdese que, en terminología de Oracle, un ejemplar es la combinación de procesos en segundo plano y áreas de memoria). Esta característica permite a Oracle ejecutarse en arquitecturas de hardware agrupadas y MPP (disco compartido y sin compartimiento). Esta característica se denominó servidor paralelo de Oracle (Oracle Parallel Server) en versiones anteriores de Oracle. La capacidad de agrupar varios nodos tiene importantes ventajas en la dimensionabilidad y disponibilidad que son útiles en entornos OLTP y de almacén de datos.

Las ventajas de dimensionabilidad de la característica son obvias, puesto que más nodos significa más potencia de procesamiento. Oracle optimiza más todavía el uso del hardware a través de las características tales como las reuniones por afinidad y por particiones.

Las agrupaciones de aplicaciones reales de Oracle9i también se pueden utilizar para lograr una alta disponibilidad. Si un nodo falla, los restantes todavía están disponibles para que la aplicación acceda a la base de datos. Las instancias restantes automáticamente retroceden las transacciones sin compromiso que están siendo procesadas en el nodo que falló con el fin de evitar un bloqueo de la actividad en el resto de nodos.

La ejecución de varias instancias en la misma base de datos da lugar a varios temas técnicos que no existen en un único ejemplar. Mientras que algunas veces es posible dividir una aplicación entre los nodos de forma que los nodos raramente accedan a los mismos datos, siempre hay posibilidad de solapamiento, que afecta a la gestión de los bloqueos y de la caché. Para solucionar esto Oracle soporta un gestor de bloqueos distribuidos y la característica *mezcla de cachés*, que permite a los bloques de datos fluir directamente entre las cachés de distintos ejemplares mediante el uso de la interconexión, sin ser escritas a disco.

25.7. RÉPLICAS, DISTRIBUCIÓN Y DATOS EXTERNOS

Oracle proporciona soporte para las réplicas y transacciones distribuidas con compromiso de dos fases.

25.7.1. Réplica

Oracle soporta varios tipos de réplica (véase el Apartado 19.2.1 para una introducción a las réplicas). En su forma más sencilla los datos en un sitio maestro se duplican en otros sitios en forma de instantáneas (el término «instantánea» en este contexto no se debería confundir con el concepto de instantánea consistente en lectura en el contexto del modelo de concurrencia). Una instantánea no tiene que contener todos los datos maestros (puede, por ejemplo, excluir ciertas columnas de una tabla

por razones de seguridad). Oracle soporta dos tipos de instantáneas: sólo lectura y actualizable. Una instantánea actualizable se puede modificar en el sitio esclavo y las modificaciones se propagan hasta la tabla maestra. Sin embargo, las instantáneas sólo lectura permiten un rango más amplio de definiciones de instantánea. Por ejemplo, una instantánea de sólo lectura se puede definir en términos de conjuntos de operaciones sobre tablas en el sitio maestro. Oracle también soporta varios sitios maestros para los mismos datos, donde todos los sitios maestros actúan como pares. Se puede actualizar una tabla replicada en cualquiera de los sitios maestro y la actualización se propaga al resto de sitios. Las actualizaciones se pueden propagar de forma asíncrona o síncrónica.

Para la réplica asíncrona la información de actualización se envía mediante procesos por lotes al resto de sitios maestros y entonces se aplican. Puesto que los mismos datos podrían estar sujetos a modificaciones conflictivas en sitios diferentes, se podría necesitar una resolución del conflicto basada en algunas reglas del negocio. Oracle proporciona una serie de métodos de resolución de conflictos incorporados y permite a los usuarios escribir el suyo propio si fuera necesario.

Con la réplica asíncrona, una actualización de un sitio maestro se propaga de forma inmediata al resto de sitios. Si falla la transacción de actualización en cualquier sitio maestro, la actualización se deshace en todos los sitios.

25.7.2. Bases de datos distribuidas

Oracle soporta consultas y transacciones sobre varias bases de datos en distintos sistemas. Con el uso de pasarelas los sistemas remotos pueden incluir bases de datos que no sean de Oracle. Oracle tiene capacidades incorporadas para optimizar una consulta que incluya tablas en distintos sitios, recuperar los datos relevantes y devolver los resultados como si hubiera sido una consulta normal local. Oracle también soporta la emisión transparente de transacciones a varios sitios mediante un protocolo de compromiso en dos fases incorporado.

25.7.3. Orígenes de datos externos

Oracle tiene varios mecanismos para soportar orígenes de datos externos. El uso más común es el almacén de datos cuando se cargan normalmente grandes cantidades de datos desde un sistema transaccional.

*25.7.3.1. SQL*Loader*

Oracle tiene una utilidad de carga directa, SQL*Loader, que soporta cargas rápidas en paralelo de grandes cantidades de datos desde archivos externos. Soporta una serie de formatos de datos y puede ejecutar varias operaciones de filtrado sobre los datos que se están cargando.

25.7.3.2. Tablas externas

Oracle permite hacer referencia a los orígenes de datos externos, tales como archivos planos, en la cláusula **from** de una consulta como si fueran tablas normales. Una tabla externa se define mediante metadatos que describen los tipos de columna Oracle y la correspondencia entre los datos externos y dichas columnas. También es necesario un controlador de acceso para acceder a los datos externos. Oracle proporciona un controlador predeterminado para archivos planos.

La característica de tabla externa tiene el objetivo principal de operaciones de extracción, transformación y carga (ETL) en un entorno de almacén de datos. Los datos se pueden cargar en el almacén de datos desde un archivo plano mediante el uso de

create table *tabla* **as**
select … from < tabla externa >
where …

Mediante la agregación de operaciones sobre los datos en la lista **select** o cláusula **where**, se pueden realizar transformaciones y filtrados como parte de la misma instrucción SQL. Puesto que estas operaciones se pueden expresar en SQL nativo o en funciones escritas en PL/SQL o Java, la característica de tabla externa proporciona un mecanismo potente para expresar todas las clases de operaciones de transformación y filtrado de los datos. Para la dimensionabilidad, se puede realizar en paralelo el acceso a la tabla externa mediante la característica de ejecución en paralelo de Oracle.

25.8. HERRAMIENTAS DE GESTIÓN DE BASES DE DATOS

Oracle proporciona a los usuarios una serie de herramientas para la gestión del sistema y desarrollo de aplicaciones.

25.8.1. Gestor corporativo de Oracle

El gestor corporativo de Oracle (Oracle Enterprise Manager) es la principal característica de Oracle para la gestión de sistemas de bases de datos. Proporciona una interfaz de usuario gráfica (GUI) sencilla de utilizar y una serie de asistentes para la administración del esquema, de la seguridad, de los ejemplares, del almacenamiento y de la planificación de tareas. También proporciona la supervisión del rendimiento y herramientas para ayudar a los administradores a ajustar la aplicación SQL, rutas de acceso y parámetros de almacenamiento

de ejemplares y datos. Por ejemplo, incluye un asistente que puede sugerir los índices que son los más efectivos de crear bajo una carga de trabajo dada.

25.8.2. Gestión de los recursos de la base de datos

Un administrador de la base de datos necesita poder controlar cómo se divide la potencia de procesamiento entre los usuarios y grupos de usuarios. Algunos grupos pueden ejecutar consultas interactivas donde el tiempo de respuesta es crítico; otros pueden ejecutar informes largos que se pueden ejecutar como tareas de procesos por lotes en segundo plano cuando la carga del sistema sea baja. También es importante poder evitar que un usuario envíe inadvertidamente una consulta ad hoc extremadamente costosa que retrasará demasiado al resto.

La característica de gestión de los recursos de la base de datos de Oracle permite al administrador de la base de datos dividir los usuarios entre grupos consumidores de recursos con distintas prioridades y propiedades. Por ejemplo, un grupo de usuarios interactivos de alta prioridad pueden tener garantizado al menos un 60 por ciento de UCP. El resto, más alguna parte del 60 por ciento no utilizado por el grupo de alta prioridad, se asignaría entre los grupos de consumidores de recursos con baja prioridad. Un grupo de realmente baja prioridad podría tener asignado un 0 por ciento, lo que significaría que las consultas enviadas por este grupo se ejecutarían solamente cuando hubiera disponibles ciclos de CPU no utilizados. Se pueden establecer para cada grupo límites para el grado de paralelismos para la ejecución en paralelo. El administrador de la base de datos también puede establecer límites de tiempo sobre cuánto tiempo máximo de ejecución se permite a una instrucción SQL. Cuando un usuario envía una instrucción, el gestor de recursos estima cuánto tiempo tardaría en ejecutarse y devuelve un error si la instrucción viola el límite. El gestor de recursos también puede limitar el número de sesiones de usuario que se pueden activar simultáneamente para cada grupo de consumidores de recursos.

NOTAS BIBLIOGRÁFICAS

Se puede encontrar información actualizada, incluyendo documentación, sobre productos Oracle en http://www.oracle.com y http://technet.oracle.com. La indexación extensible en Oracle8i se describe en Srinivasan et al. [2000b], mientras que Srinivasan et al. [2000a] describe las tablas organizadas con índices en Oracle8i. Banerjee et al. [2000] describe soporte XML en Oracle8i. Bello et al. [1998] describe las vistas materializadas en Oracle. Antoshenkov [1995] describe la técnica de compresión de mapas de bits alineadas por bytes utilizada en Oracle; véase también Johnson [1999b]. Oracle Parallel Server se describe en Bamford et al. [1998]. La recuperación en Oracle viene descrita por Joshi et al. [1998] y Lahiri et al. [2001]. La mensajería y las colas en Oracle se describen en Gawlick [1998].

PARTE

VIII

ESTUDIO DE CASOS

Esta parte describe cómo los distintos sistemas de bases de datos integran los diferentes conceptos descritos anteriormente en el libro. Específicamente, en los Capítulos 25, 26 y 27 se cubren tres sistemas de bases de datos ampliamente utilizados: DB2 de IBM, Oracle y SQL Server de Microsoft. Cada uno de estos capítulos muestra características únicas de cada sistema de bases de datos: herramientas, variaciones y extensiones de SQL y la arquitectura del sistema, incluyendo organización del almacenamiento, procesamiento de consultas, control de concurrencia y recuperación y réplicas.

Los capítulos cubren solamente los aspectos clave de los productos de bases de datos que describen, y por consiguiente, no se deberían utilizar como una descripción completa del producto. Además, puesto que los productos se mejoran periódicamente, los detalles del producto pueden cambiar. Cuando se utiliza una versión particular del producto hay que asegurarse de consultar los manuales de usuario para los detalles específicos.

Hay que considerar que los capítulos en esta parte utilizan terminología industrial en lugar de académica. Por ejemplo, se utiliza tabla en lugar de relación, fila en lugar de tupla y columna en lugar de atributo.

CAPÍTULO

26

DB2 DE IBM

SRIRAM PADMANABHAN
IBM T. J. WATSON RESEARCH CENTER

La familia de productos DB2 Universal Database de IBM consiste en servidores de bases de datos y un conjunto de productos relacionados. DB2 Universal Database Server está disponible en muchas plataformas hardware y sistemas operativos, abarcando desde *mainframes* (grandes ordenadores centrales) y grandes servidores a estaciones de trabajo e incluso a pequeños dispositivos de bolsillo. Se ejecuta en una serie de sistemas operativos IBM y de otras marcas. Everyplace Edition soporta sistemas operativos tales como PalmOS, Windows CE y otros. Las aplicaciones pueden migrar fácilmente desde las plataformas de gama baja a servidores de gama alta. Además del motor del núcleo de la base de datos, la familia DB2 consta también de varios otros productos que proporcionan herramientas, administración, réplicas, acceso a datos distribuido, acceso a datos generalizados, OLAP y otras muchas características. En la Figura 26.1 se describen los diferentes productos en la familia.

El origen de DB2 se remonta al proyecto System R en el Centro de Investigación de Almadén (Almaden Research Center) de IBM (entonces denominado Laboratorio de investigación de San José de IBM, IBM San Jose Research Laboratory). El primer producto DB2 se lanzó en 1984 sobre la plataforma mainframe de IBM. Fue seguido por otras versiones para otras plataformas. IBM ha mejorado continuamente el producto DB2 en áreas tales como procesamiento de transacciones (registro histórico de escritura anticipada y los algoritmos de recuperación ARIES), procesamiento y optimización de consultas (proyecto de investigación Starburst), procesamiento en paralelo (DB2 Parallel Edition), soporte para bases de datos activas (restricciones y disparadores) y soporte relacional orientado a objetos aprovechando las innovaciones de su división de investigación. Se pueden ver las notas bibliográficas para referencias que proporcionan más detalles.

El motor de la base de datos DB2 está disponible en cuatro bases de código diferentes: (1) OS/390, (2) VM, (3) AS/400 y (4) resto de plataformas. Los elementos comunes en todas estas bases de código son interfaces externas (en concreto el lenguaje de definición de datos (LDD) y SQL) y herramientas básicas tales como administración. Sin embargo, existen diferencias como resultado de diferentes historias de desarrollo para las bases de código. El resto de este capítulo se enfocará en DB2 Universal Database System de plataformas Unix, Windows y OS/2. Se reseñarán las características específicas de interés en otros sistemas DB2 cuando se considere apropiado.

Servidores de bases de datos
- DB2 UDB para Unix, Windows, OS/2, Linux
- DB2 UDB para OS/390
- DB2 UDB para AS/400
- DB2 para VM/VSE

Desarrollo de aplicaciones
- VisualAge para Java, C++
- VisualAge Generator
- DB2 Forms para OS/390
- Lotus Approach

Inteligencia de negocios
- DB2 OLAP Server
- DB2 Intelligent Miner
- DB2 Spatial Extender
- DB2 Warehouse Manager
- QMF para OS/390

Herramientas para el negocio electrónico (E-Business)
- DB2 Net Search Extender
- DB2 XML Extender
- Net.Data
- DB2 para Websphere

Integración de datos
- DB2 DataJoiner
- DataLinks
- Data Replication Services
- DB2 Connect

Gestión de contenidos
- Content Manager
- Content Manager VideoCharger

Herramientas de gestión de bases de datos
- DB2 Control Center
- DB2 Admin Tools para OS/390
- DB2 Buffer Pool Tool
- DB2 Estimator para OS/390
- DB2 Performance Monitor
- DB2 Visual Explain
- DB2 Query Patroller
- etc.

Acceso móvil a datos
- DB2 EveryPlace
- DB2 Satellite Edition

Multimedia
- DB2 ObjectRelational Extenders
- Digital Library

FIGURA 26.1. Familia de productos DB2.

26.1. HERRAMIENTAS PARA EL DISEÑO DE BASES DE DATOS Y LA CONSULTA

La mayor parte del diseño de base de datos y herramientas CASE se puede utilizar para diseñar una base de datos DB2. En particular, las herramientas de modelado de datos líderes tales como ERWin y Rational Rose permiten al diseñador generar sintaxis del LDD específica de DB2. Por ejemplo, la herramienta UML Data Modeler de Rational Rose puede generar instrucciones **create distinct type** del LDD específico de DB2 para tipos definidos por el usuario y utilizarlos posteriormente en definiciones de columnas. La mayor parte de herramientas de diseño también soportan una característica de ingeniería inversa que lee las tablas del catalogo DB2 y construye un diseño lógico para manipulaciones adicionales. Las herramientas pueden generar restricciones e índices.

DB2 proporciona constructoras SQL para soportar muchas características de bases de datos lógicas, tales como restricciones, disparadores y recursión. De igual forma, DB2 soporta ciertas características de bases de datos físicas tales como espacios de tablas, colas de memoria intermedia y división mediante el uso de instrucciones SQL. La herramienta Control Center GUI para DB2 permite a los diseñadores o administradores emitir la instrucción LDD apropiada para estas características. Otra herramienta permite al administrador obtener un conjunto completo de instrucciones LDD para una base de datos incluyendo espacios de tablas, tablas, índices, restricciones, disparadores y funciones que crean una réplica exacta del esquema de la base de datos para verificación o réplica.

Para el análisis de datos DB2 proporciona soporte OLAP mediante el servidor DB2 OLAP. El servidor DB2 OLAP puede crear un puesto de datos multidimensional desde una base de datos DB2 subyacente para su análisis. El motor OLAP del producto Essbase se utiliza en el servidor DB2 OLAP. DB2 también soporta otros motores OLAP de fabricantes tales como Microstrategy y Cognos. En particular, DB2 proporciona soporte nativo para las instrucciones **cube by** y **rollup** para generar cubos agregados, así como agregados junto a una o más jerarquías en el motor de la base de datos. Intelligent Miner y otros productos de minería de datos se pueden utilizar para análisis más profundos y complejos en datos DB2.

DB2 para OS/390 tiene un conjunto muy grande de herramientas. QMF es una herramienta ampliamente utilizada para generar consultas ad hoc e integrarlas en aplicaciones.

26.2. VARIACIONES Y EXTENSIONES DE SQL

DB2 soporta un amplio conjunto de características SQL para varios aspectos del procesamiento de la base de datos. Muchas de las características y sintaxis de DB2 han proporcionado la base de los estándares en SQL-92 y SQL:1999. Este apartado resalta las características relacionales orientadas a objetos en DB2 UDB versión 7. El lector puede encontrar referencias para completar la descripción del soporte para SQL de DB2 de IBM, así como extensiones al soporte XML en las notas bibliográficas.

26.2.1. Soporte para tipos de datos

DB2 soporta tipos de datos definidos por el usuario. Los usuarios pueden definir tipos de datos *distintos* o *estructurados*. Los tipos de datos distintos se basan en los tipos de datos incorporados en DB2. Sin embargo, los usuarios pueden definir semánticas adicionales o alternativas para estos nuevos tipos. Por ejemplo, el usuario puede definir un tipo de datos distinto denominado EURO como

create distinct type EURO **as decimal**(9,2).

Por consiguiente, el usuario puede crear un campo (por ejemplo, PRECIO) en una tabla cuyo tipo sea EURO. Las consultas pueden utilizar el campo con este tipo en los predicados como en el siguiente ejemplo:

```
select Producto
from Ventas_Europa
where Precio > EURO(1000)
```

Los tipos de datos estructurados son objetos complejos que normalmente se componen de dos o más atributos. El siguiente código declara un tipo de datos estructurado denominado t_departmento:

```
create type t_departmento as
    (nombredept varchar(32),
    directordept varchar(32),
    número integer)
mode db2sql
```

Los tipos estructurados se pueden utilizar para definir tablas con tipos.

create table dept **of** t_departmento

Con el LDD un diseñador de sistema puede crear una jerarquía de tipos y tablas en la jerarquía que puede heredar métodos específicos y privilegios. Los tipos estructurados también se pueden utilizar para definir atributos anidados dentro de una columna o tabla.

26.2.2. Funciones y métodos definidos por el usuario

Otra característica importante es que los usuarios pueden definir sus propias funciones y métodos. Estas funciones se pueden posteriormente incluir en instrucciones y consultas SQL. Las funciones pueden generar escalares (único atributo) o tablas (fila multiatributo) como resultado. Los usuarios pueden registrar funciones (escalares o de tablas) mediante el uso de la instrucción **create function**. Pueden escribir las funciones en lenguajes de programación comunes tales como C y Java o lenguajes de guiones tales como Rexx y Perl. Las funciones definidas por el usuario (FDU) pueden operar en los modos separado (*fenced*) y compartido (*unfenced*). En el modo separado las funciones se ejecutan mediante una hebra separada en su propio espacio de dirección. En el modo compartido se permite al agente de procesamiento de la base de datos ejecutar la función en el espacio de direcciones del servidor. Las FDU pueden definir un área de trabajo donde pueden mantener variables locales y estáticas en invocaciones diferentes.

Otra característica son los métodos asociados con los objetos, los cuales definen el comportamiento de los objetos. Los métodos están asociados con tipos de datos estructurados particulares y se registran mediante el uso de la instrucción **create method**.

26.2.3. Objetos de gran tamaño

Las nuevas aplicaciones de las bases de datos requieren la manipulación de texto, imágenes, vídeo y otros tipos de datos típicos que son bastante grandes. DB2 soporta estos requisitos proporcionando tres tipos de objetos de gran tamaño (LOB, Large Object) distintos. Cada LOB puede ser de hasta 2 gigabytes de tamaño. Los objetos de gran tamaño en DB2 son (1) objetos en binario de gran tamaño (Binary Large Objetcs, BLOBs), (2) objetos de caracteres de gran tamaño de un único byte (Character Large Objects, CLOBs) y (3) objetos de caracteres de gran tamaño de dos bytes (Double Byte Character Large Objects, DBCLOBs). DB2 organiza estos LOBs como objetos separados, con cada fila en la tabla manteniendo punteros a sus LOBs correspondientes. Los usuarios pueden registrar UDFs que manipulen estos LOBs según los requisitos de la aplicación.

26.2.4. Soporte para XML

DB2 integra el soporte para XML en el servidor mediante el uso de XML extendido. XML extendido puede extraer elementos y atributos XML en columnas de datos relacionales y mejorar SQL y el poder de indexación de DB2. De forma alternativa también puede almacenar y recuperar documentos XML como una única columna en una tabla. Puede indexar y proporcionar capacidades de búsqueda orientada a texto en esta columna XML. El extensor proporciona una serie de funciones incorporadas y APIs para la composición, inserción, actualización y búsqueda en documentos XML. Es probable que se integren pronto nuevas características tales como la exposición de los datos DB2 como servicio Web mediante el protocolo SOAP y soporte de consultas XML.

26.2.5. Extensiones de índices y restricciones

Una característica reciente de DB2 proporciona un constructor **create index extension** que ayuda a crear índices sobre atributos con tipos de datos estructurados mediante la generación de claves a partir de los tipos de datos estructurados. Por ejemplo, un usuario puede crear un índice en un atributo cuyo tipo es t_departamento (definido en la sección 26.2.1) mediante la generación de claves con el nombre departamento. El extensor espacial de DB2 utiliza el método de extensión de índice para crear índices sobre los datos espaciales. DB2 también proporciona un rico conjunto de características de verificación de restricciones para imponer la semántica de los objetos tales como unicidad, validez y herencia.

26.3. ALMACENAMIENTO E INDEXACIÓN

IBM DB2 proporciona una serie de características para el almacenamiento e indexación de datos.

26.3.1. Arquitectura de almacenamiento

DB2 proporciona abstracciones de almacenamiento para gestionar tablas de base de datos lógicas en entornos multinodo (paralelo) y multidisco. Se pueden definir grupos de nodos para soportar la división de la tabla en conjuntos especificados de nodos en un sistema multinodo. Esto permite flexibilidad al asignar particiones de tabla a nodos diferentes en un sistema. Por ejemplo, las tablas de gran tamaño se pueden dividir entre todos los nodos en un sistema mientras que las tablas pequeñas pueden residir en un único nodo. Las tablas se dividen por asociación entre los nodos en el grupo de nodos utilizando un subconjunto de sus atributos (clave de división).

Dentro de un nodo, DB2 utiliza espacios de tablas para reorganizar la tabla. Un espacio de tablas consiste en uno o más contenedores que son referencias a directorios, dispositivos o archivos. Un espacio de tablas pue-

de contener cero o más objetos de base de datos tales como tablas, índices o LOBs. La Figura 26.2 ilustra estos conceptos. En esta figura se definen dos espacios de tablas para un grupo de nodos. Al espacio de tablas RECHUMANOS se asignan cuatro contenedores mientras que el espacio de tablas PLAN tiene solamente un contenedor. Las tablas EMPLEADO y DEPARTAMENTO están en el espacio de tablas RECHUMANOS mientras que la tabla PROYECTO está en el espacio de tablas PLAN. La distribución de datos asigna fragmentos (extensiones) de las tablas EMPLEADO y DEPARTAMENTO a los contenedores del espacio de tablas RECHUMANOS. DB2 permite al administrador crear tanto espacios de tablas gestionados por el sistema como por el SGBD. Los espacios de tablas gestionados por el sistema (System-managed spaces, SMSs) son directorios o sistemas de archivo que mantiene el sistema operativo subyacente. En un SMS, DB2 crea objetos archivo en los directorios y asigna datos a cada uno de los archivos. Los espacios de tablas gestionados por el SGBD (Data Managed Spaces, DMSs) son dispositivos en bruto o archivos preasignados que son controlados por DB2. El tamaño de estos contenedores nunca puede crecer o disminuir. DB2 crea mapas de asignación y gestiona el espacio de tablas DMS. En ambos casos la unidad de espacio de almacenamiento es una extensión de páginas. El administrador puede elegir el tamaño de la extensión para un espacio de tabla. DB2 soporta la distribución en distintos contenedores. Por ejemplo, cuando se insertan los datos en una tabla recientemente creada, DB2 asigna la primera extensión a un contenedor. Una vez que la extensión está llena asigna los siguientes datos al siguiente contenedor por turnos rotatorios. La distribución proporciona dos ventajas significativas: E/S paralela y balance de carga. DB2 también soporta la preextracción y escrituras asíncronas mediante el uso de hebras separadas. El componente de gestión de datos de DB2 desencadena la preextracción de páginas de datos y de índices según los patrones de acceso de las consultas. Por ejemplo, una exploración de una tabla siempre desencadena la preextracción de páginas de datos. La exploración del índice puede desencadenar la preextracción de páginas de índices así como las páginas de datos si se está accediendo de una forma agrupada. El número de preextracciones concurrentes así como el tamaño de la preextracción son parámetros configurables que se necesitan iniciar según el número de discos o contenedores en el espacio de tablas.

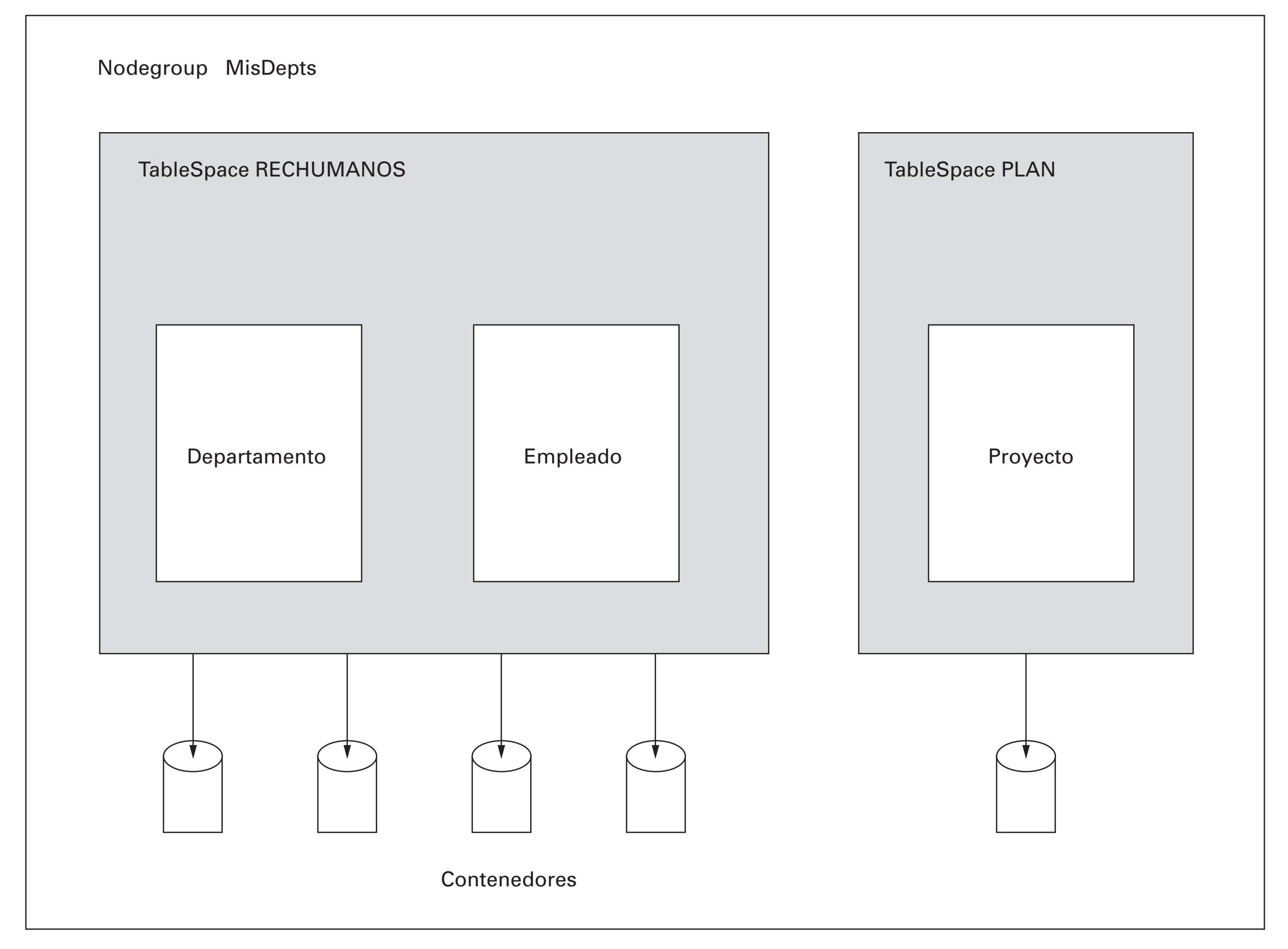


FIGURA 26.2. Espacios de tablas y contenedores en DB2.

26.3.2. Tablas, registros e índices

DB2 organiza los datos relacionales como registros en las páginas. La Figura 26.3 muestra la vista lógica de una tabla y un índice asociado. La tabla consiste en un conjunto de páginas. Cada página consiste en un conjunto de registros (tanto registros de datos del usuario como registros especiales del sistema). La página cero de la tabla contiene registros del sistema especiales sobre la tabla y su estado. DB2 utiliza un registro del mapa de espacio denominado registro de control de espacio libre (Free Space Control Record, FSCR) para encontrar el espacio libre en la tabla. El registro FSCR normalmente contiene un mapa de espacio de 500 páginas. Una entrada FSCR consiste en unos pocos bits que proporcionan una indicación aproximada del porcentaje de espacio libre en la página; por ejemplo, con dos bits, el patrón de bits 11 indicaría que la mayor parte de la página puede estar libre mientras que 01 indicaría que alrededor del 25 por ciento del espacio está libre. Para reducir el coste de actualización, las entradas no siempre se actualizan con el uso del espacio real, por lo que el código de inserción o actualización debe validar las entradas FSCR realizando una verificación física del espacio disponible en una página.

Los índices se organizan como páginas que contienen registros índice y punteros a páginas hijas y hermanas. DB2 proporciona soporte para los mecanismos de índices de árbol B^+. El índice de árbol B^+ contiene páginas internas y páginas hoja. Los índices tienen punteros bidimensionales en el nivel hoja para soportar exploraciones hacia delante y atrás. Las páginas hoja contienen entradas de índice que apuntan a los registros de la tabla. Cada registro de una tabla tiene un identificador de registro único (Register ID, RID) construido a partir de su identificador de página y de ranura en la página (la estructura de páginas con ranuras se describe en breve). Se puede definir un índice como los índices de agrupación de la tabla. Si se define este índice, los registros de datos se mantienen en un orden de agrupamiento orientado a la página según las claves del índice.

Los índices DB2 pueden almacenar columnas extra junto con los identificadores de registro en el nivel de las hojas de los índices. Por ejemplo,

create unique index I1 **on** T1 (C1) **include** (C2)

especifica que C2 se va a incluir como una columna extra en un índice sobre la columna C1. Las columnas incluidas permiten a DB2 utilizar las técnicas de procesamiento de la consulta «sólo con el índice» (evitando la lectura del registro real) para consultas que utilizan las columnas incluidas, lo que no sería posible en otro caso (el procesamiento de la consulta sólo con el índice se describirá con más detalle en breve). Se pueden utilizar directrices adicionales tales como MINPCUSED y PCTFREE para controlar la unión de páginas de índices y su asignación de espacio inicial durante la carga masiva.

La Figura 26.4 muestra el formato de datos típico en DB2. Cada página de datos contiene una cabecera y un directorio de ranuras. El directorio de ranuras es un array de 255 entradas que apuntan a los desplazamientos de los registros en la página. La figura muestra que el número de página 473 contiene el registro cero en el desplazamiento 3.800 y el registro 2 en el desplazamiento 3.400. La página 1056 contiene un registro 1 en el desplazamiento 3.700, que es realmente un puntero hacia delante al registro <473,2>. Por ello el registro <473, 2 > es un registro de desbordamiento que fue creado por una operación de actualización del registro < 1056, 1 > original. DB2 soporta distintos tamaños de página tales como 4 KB, 8 KB, 16 KB y 32 KB. Sin embargo, cada página puede contener solamente 255 registros de usuario. Los tamaños de página mayores son útiles en aplicaciones tales como almacén de datos donde la tabla contiene muchas columnas. Los tamaños de página menores son útiles para datos operacionales con frecuentes actualizaciones.

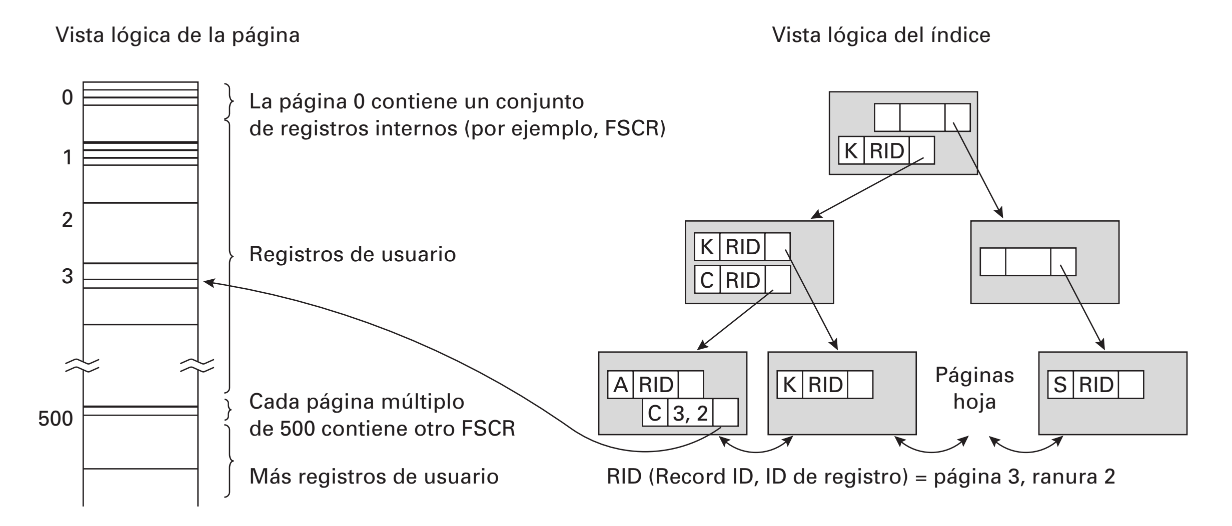


FIGURA 26.3. Vista lógica de las tablas e índices en DB2.

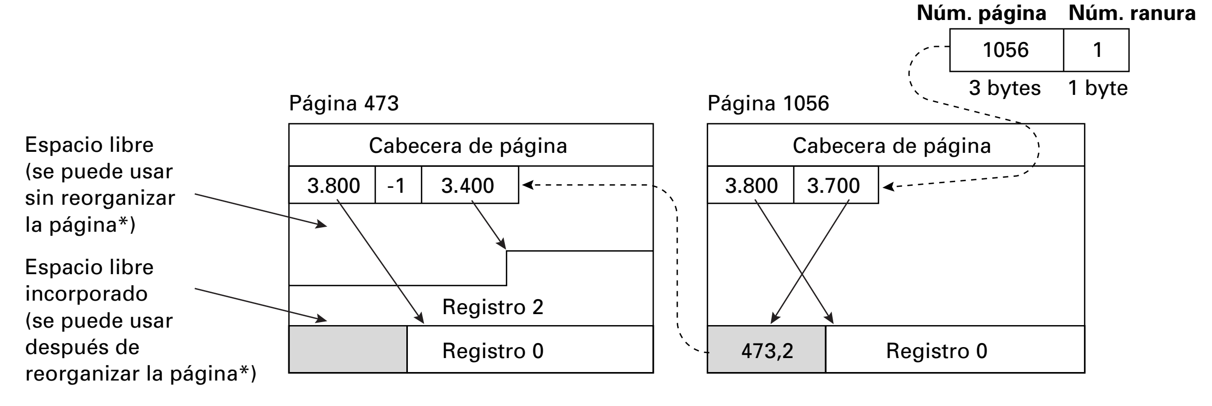


FIGURA 26.4. Diseño de las páginas de datos y de los registros en DB2.

26.4. PROCESAMIENTO Y OPTIMIZACIÓN DE CONSULTAS

El compilador de consultas de DB2 transforma las consultas en un árbol de operaciones. DB2 utiliza entonces el árbol de operadores de la consulta en tiempo de ejecución para el procesamiento. DB2 soporta un rico conjunto de operadores de consulta que permiten considerar mejores estrategias de procesamiento y proporcionan flexibilidad en la ejecución de consultas complejas.

Las Figuras 26.5 y 26.6 muestran una consulta y su plan de consulta asociado. Se trata de una consulta compleja representativa (consulta 5) de la prueba TPC-H y contiene varias reuniones y agregaciones. El plan de consulta en este ejemplo es bastante simple, puesto que solamente se definen pocos índices y no están disponibles para esta consulta estructuras auxiliares como las vistas materializadas. DB2 proporciona varias características de explicación del plan incluyendo una potente característica visual en el centro de control que puede ayudar a los usuarios a comprender los detalles del plan de ejecución de la consulta. El plan de consulta en la figura está basado en la explicación visual de la consulta. La explicación visual permite al usuario comprender los costes y otras propiedades relevantes de las distintas operaciones de un plan de consulta.

DB2 transforma todas las consultas e instrucciones SQL, sin importar lo complejas que sean, en un árbol de consulta. La base u operadores hoja del árbol de consulta manipulan los registros en tablas de base de datos. Estas operaciones también se denominan métodos de acceso. Las operaciones intermedias del árbol incluyen operaciones del álgebra relacional tales como reuniones, operaciones de conjuntos y agregaciones. La raíz del árbol produce los resultados de la consulta o instrucción SQL.

26.4.1. Métodos de acceso

DB2 soporta un conjunto detallado de métodos de acceso sobre tablas relacionales, incluyendo.

- **Exploración de tabla.** Con este método, el más básico, se accede a todos los registros en la tabla página por página.
- **Exploración de índice.** DB2 utiliza un índice para seleccionar los registros específicos que satisfacen la consulta. Accede a los registros utilizando los RIDs en el índice. DB2 detecta las posibilidades de la preextracción de las páginas de datos cuando observa un patrón de acceso secuencial.
- **Sólo con el índice.** Este tipo de exploración se utiliza cuando el índice contiene todos los atributos que requiere la consulta. Por ello es suficiente una exploración de las entradas de índice y no hay necesidad de extraer los registros. La técnica sólo con el índice es normalmente una buena elección desde el punto de vista del rendimiento.
- **Lista de preextracción.** Este método de acceso es una buena elección para una exploración de índices no agrupada con un número significativo de RIDs. DB2 recoge los RIDs de los registros relevantes utilizando una exploración de índices, entonces ordena los RIDs por el número de página y finalmente realiza una extracción de los registros de forma ordenada desde las páginas de datos. El acceso ordenado cambia el patrón E/S de aleatorio a secuencial y también ofrece posibilidades de preextracción.
- **Conjunción de índices.** DB2 utiliza este método cuando determina que se puede utilizar más de un

```
–'TPCD Local Supplier Volume Query (Q5)';
select N_NAME,
    sum(L_EXTENDEDPRICE*(1-L_DISCOUNT)) as REVENUE
from TPCD.CUSTOMER, TPCD.ORDERS, TPCD.LINEITEM,
    TPCD.SUPPLIER, TPCD.NATION, TPCD.REGION
where C_CUSTKEY = O_CUSTKEY
    and O_ORDERKEY = L_ORDERKEY
    and L_SUPPKEY = S_SUPPKEY
    and C_NATIONKEY = S_NATIONKEY
    and S_NATIONKEY = N_NATIONKEY
    and N_REGIONKEY = R_REGIONKEY
    and R_NAME = 'MIDDLE EAST'
    and O_ORDERDATE >= date('1995-01-01')
    and O_ORDERDATE < date('1995-01-01')+ 1 year
group by N_NAME
order by REVENUE DESC;
```

FIGURA 26.5. Consulta SQL

índice para restringir el número de registros satisfactorios en una tabla base. Procesa el índice más selectivo para generar una lista de RIDs. Entonces procesa el siguiente índice selectivo para devolver los RIDs que encuentra. Un RID requiere más procesamiento solamente si está presente en la intersección (operación AND) de los resultados de la exploración del índice. El resultado de una operación AND del índice es una pequeña lista del RIDs que se utilizan para extraer los registros correspondientes desde la tabla base.

- Disyunción de índices. Esta estrategia es una buena elección si se pueden utilizar dos o más índices para satisfacer los predicados de la consulta que se combinan utilizando la operación OR. DB2 elimina los RIDs duplicados realizando una ordenación y entonces extrae el conjunto de registros resultante.

DB2 normalmente envía todos los predicados de selección y proyección de una consulta a los métodos de acceso. Además DB2 envía ciertas operaciones tales

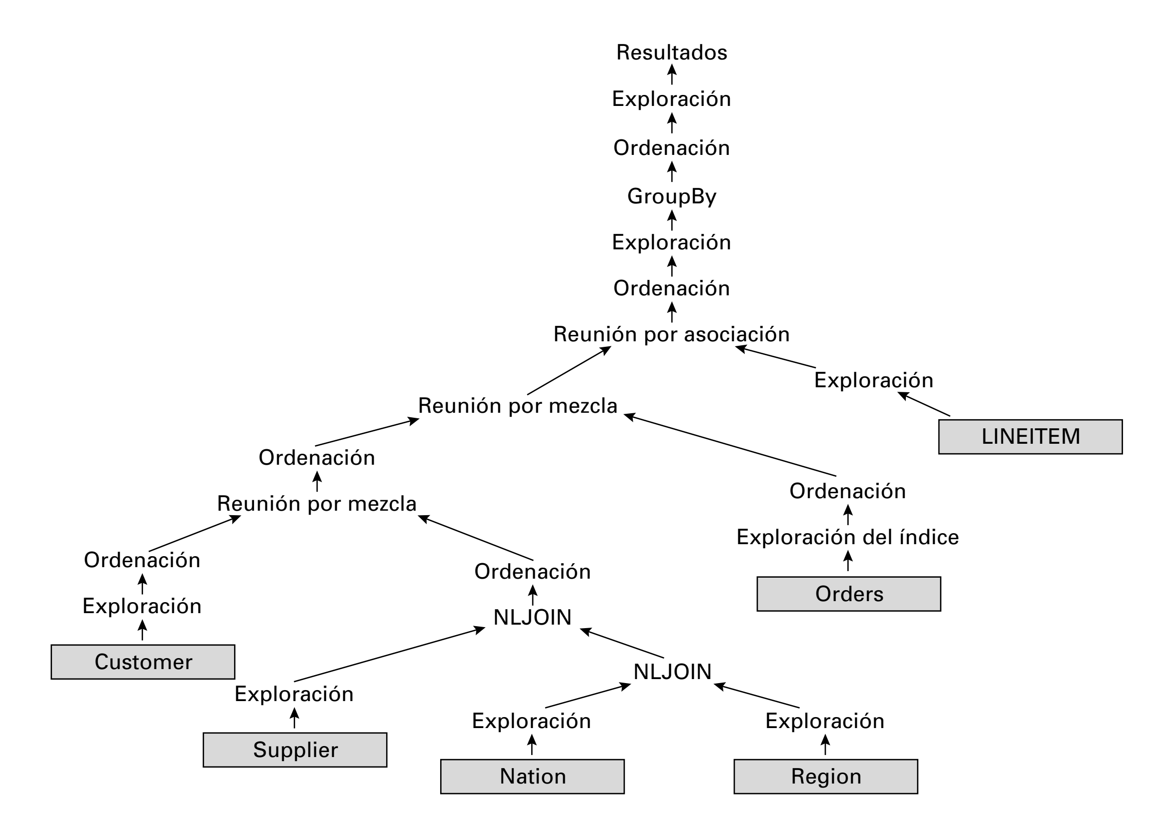


FIGURA 26.6. Plan de consulta DB2 (explicación gráfica).

como la ordenación y la agregación, siempre que es posible, con el fin de reducir el coste.

26.4.2. Operaciones de reunión, agregación y de conjuntos

DB2 soporta una serie de técnicas para las operaciones de reunión, agregación y de conjuntos. Para la reunión DB2 puede elegir entre técnicas de bucles anidados, ordenación-mezcla y de asociación. Para describir las operaciones binarias de reuniones y de conjuntos se utilizará la notación de las tablas externas e internas para distinguir los dos flujos de entrada. La técnica de bucles anidados es útil si la tabla interna es muy pequeña o se puede acceder utilizando un índice sobre un predicado de reunión. Las técnicas de reunión de ordenación-mezcla y reunión por asociación son útiles para reuniones que involucran tablas internas y externas grandes. DB2 implementa las operaciones de conjuntos mediante el uso de técnicas de ordenación y mezcla. La técnica de mezcla elimina los duplicados en el caso de la unión mientras que los no duplicados se eliminan en el caso de intersección. DB2 también soporta operaciones de reunión externa de todas las clases.

DB2 procesa las operaciones de agregación en modo impaciente o de envío siempre que sea posible. Por ejemplo, puede realizar la agregación mientras que ordena la entrada de la agregación en el grupo por columnas. Los algoritmos de reunión y agregación aprovechan el procesamiento superescalar en CPUs modernas utilizando técnicas orientadas a bloques y conscientes de la caché de memoria.

26.4.3. Soporte para el procesamiento de SQL complejo

Uno de los aspectos más importantes de DB2 es que utiliza la infraestructura de procesamiento de la consulta de forma extensible para soportar operaciones SQL complejas. Las operaciones SQL complejas incluyen soporte para subconsultas profundamente anidadas y correlacionadas, así como restricciones, integridad referencial y disparadores. Las restricciones y comprobaciones de integridad se construyen como operaciones del árbol de consulta a partir de las instrucciones SQL de inserción, borrado o actualización. Mediante la ejecución de la mayoría de las acciones de verificación de restricciones y desencadenamiento como parte del plan de consulta DB2 puede proporcionar una mejor eficiencia y dimensionabilidad. DB2 también soporta el mantenimiento de vistas materializadas mediante el uso de disparadores incorporados.

26.4.4. Procesamiento de consultas en multiprocesadores

DB2 extiende el conjunto base de operaciones de consulta con primitivas de intercambio de datos y control para soportar los modos SMP, MPP y SMP por agrupaciones del procesamiento de consultas. DB2 utiliza una abstracción *tabla-cola* para el intercambio de datos entre hebras sobre distintos nodos o sobre el mismo nodo. La tabla-cola es una memoria intermedia que redirige los datos a receptores apropiados mediante el uso de métodos de difusión, uno a uno o multidifusión dirigida. Las operaciones de control crean hebras y coordinan la operación de distintos procesos y hebras.

En todos estos modos DB2 emplea un proceso coordinador para controlar las operaciones de colas y la reunión del resultado final. Los procesos de coordinación también pueden ejecutar algunas acciones globales de procesamiento de la base de datos si es necesario. Un ejemplo es la operación de agregación global para combinar los resultados de agregación local. Los subagentes o hebras esclavos ejecutan las operaciones base en uno o más modos. En el modo SMP los subagentes utilizan memoria compartida para sincronizarse entre sí cuando comparten datos. En un MPP, los mecanismos de tabla-cola proporcionan memoria intermedia y control de flujo para la sincronización entre distintos nodos durante la ejecución.

26.4.5. Optimización de consultas

El optimizador de consultas de DB2 utiliza una representación interna de la consulta, denominada Query Graph Model (QGM, modelo de grafos de consultas) con el fin de ejecutar transformaciones y optimizaciones. Después de analizar la instrucción SQL, DB2 ejecuta transformaciones semánticas sobre QGM para hacer cumplir las restricciones, integridad referencial y los disparadores. El resultado de estas transformaciones es un QGM mejorado. Seguidamente DB2 intenta ejecutar transformaciones de reescritura de la consulta que se consideran beneficiosas en la mayoría de las consultas. Se activan las reglas de reescritura, si son aplicables, para ejecutar las transformaciones requeridas. Los ejemplos de transformaciones de reescritura incluyen (1) descorrelación de subconsultas correlacionadas, (2) transformación de subconsultas en reuniones donde sea posible, (3) trasladar las operaciones group by bajo las reuniones si es aplicable y (4) reescritura de consultas para hacer uso de las vistas materializadas disponibles o «tablas resumen» (vistas materializadas utilizando la agregación).

El optimizador de consultas utiliza QGM mejorado y transformado como su entrada para la optimización. El optimizador se basa en el coste y utiliza un entorno extensible, controlado por reglas. Se puede configurar el optimizador para operar a distintos niveles de complejidad. En el nivel más alto utiliza un algoritmo de programación dinámica para considerar todas las opciones del plan de consulta y elige el plan de coste óptimo. En un nivel intermedio el optimizador no considera ciertos planes o métodos de acceso (por ejemplo, indexación) así como algunas reglas de reescritura. En el nivel

inferior de complejidad el optimizador utiliza una heurística impaciente simple para elegir un buen, aunque no necesariamente óptimo, plan de consulta. El optimizador utiliza modelos detallados de las operaciones de procesamiento de la consulta (teniendo en cuenta detalles tales como tamaño de la memoria y preextracción) para obtener estimaciones adecuadas de los costes de E/S y CPU. Depende de la estadística de los datos para estimar la cardinalidad y selectividades de las operaciones.

DB2 permite a un usuario generar histogramas de distribuciones en el nivel de las columnas y combinaciones de columnas mediante el uso de la utilidad *runstats*. Los histogramas contienen información sobre las apariciones del valor más frecuente, así como sobre las distribuciones de frecuencia basadas en los cuantiles de los atributos. El optimizador de consultas usa estas estadísticas. El optimizador genera un plan de consulta interno que considera el mejor plan de consulta y entonces convierte el plan de consulta en hebras de operadores y estructuras de datos asociados de la consulta para su ejecución mediante el motor de procesamiento de consultas.

Considérese una actualización de la forma «Aumentar en un 10 por ciento el sueldo de los empleados que ganen menos que 25.000 euros». En una versión anterior de System R el optimizador elegía el índice del sueldo para el acceso y procesaba los datos de menor a mayor (0 a 25.000). La inserción de un nuevo valor (mayor) para el sueldo significaba que el registro actualizado se revisa de nuevo a no ser que el valor del sueldo fuera mayor que 25.000. La causa de la revisión es el uso del índice del sueldo para el acceso en orden menor a mayor. Se revisaba el registro y el sueldo se aumentaba un 10 por ciento repetidamente hasta que su valor excedía a 25.000. Este error ocurrió el día de Halloween, por lo que los miembros del proyecto System R lo denominaron al error «*problema Halloween*».

DB2 soluciona el problema Halloween reconociendo esta situación en el compilador de la consulta. El optimizador genera un plan de consulta que primero materializa los RIDs de las filas implicadas antes de procesar las actualizaciones. Cada registro implicado se actualiza solamente una vez, como tenía intención la instrucción de actualización.

26.5. CONTROL DE CONCURRENCIA Y RECUPERACIÓN

DB2 soporta técnicas de control de concurrencias que proporcionan un muy alto nivel de concurrencia, acopladas con un mecanismo de recuperación avanzado que soporta una serie de características.

26.5.1. Concurrencia y aislamiento

DB2 soporta una serie de modos de control de concurrencia y aislamiento. Para el aislamiento DB2 soporta los modos lectura repetible (Repeatable Read, RR), estabilidad en lectura (Read Stability, RS), estabilidad del cursor (Cursor Stability, CS) y lectura no comprometida (Un-committed Read, UR).

La definición de lectura repetible en DB2 difiere de la del Apartado 16.8 y corresponde íntimamente con el nivel secuenciable descrito allí. Específicamente, RR en DB2 asegura que la exploración del rango encontrará el mismo conjunto de tuplas si se repiten. Si todas las transacciones siguen el protocolo RR entonces la planificación resultante será secuenciable. Los modos CS y UR son como se describen en el Apartado 16.8.

El modo de aislamiento RS bloquea solamente las filas que recupera una aplicación en una unidad de trabajo. En una exploración posterior la aplicación tiene garantizado ver todas estas filas (como RR) pero podría no ver nuevas filas que debería ver. Sin embargo, esto podría ser un compromiso aceptable para algunas aplicaciones con respecto al aislamiento RR estricto. Normalmente el nivel de aislamiento predeterminado es CS. Las aplicaciones pueden elegir el nivel de aislamiento con el que se quieren ejecutar. También, la mayoría de las aplicaciones comerciales disponibles soportan los distintos niveles de aislamiento y los usuarios pueden elegir la versión correcta de la aplicación para sus requisitos.

Los distintos modos de aislamiento se implementan mediante el uso de bloqueos. DB2 soporta bloqueos en el nivel de registros y de tablas. Mantiene una estructura de datos de bloqueo de tablas separado del resto de información de bloqueo. DB2 dimensiona el bloqueo en el nivel de registros al de tablas si el espacio disponible para posteriores bloqueos en la tabla de bloqueos se hace pequeño. DB2 implementa un bloqueo estricto de dos fases para todas las transacciones de actualización. Mantiene bloqueos de escritura y actualización hasta el momento del compromiso o retroceso.

DB2 soporta una serie de modos de bloqueo con el fin de maximizar la concurrencia. La Figura 26.7 muestra los distintos modos de bloqueo y proporciona una breve descripción del propósito de cada modo de bloqueo. Abajo se muestran brevemente algunos de los modos de bloqueo; véanse las referencias bibliográficas para mayor información sobre los modos de bloqueo. Los modos de bloqueo incluyen bloqueos intencionales en un nivel de tabla para proporcionar bloqueo de granularidad múltiple. DB2 también implementa el bloqueo de la clave siguiente para las inserciones y borrados que afecten a las exploraciones de índices del nivel de aislamiento RR para eliminar el problema de la lectura fantasma. Véanse las referencias bibliográfi-

Modo de bloqueo	Objetos	Interpretación
IN (intent none, sin intención)	Espacios de tablas, tablas	Lectura sin bloqueos de filas
IS (intent share, intentar compartir)	Espacios de tablas, tablas	Lectura con bloqueos de filas
NS (next key share, siguiente clave compartido)	Filas	Bloqueos de lectura para los niveles de aislamiento RS o CS
S (share, compartido)	Filas, tablas	Bloqueo de lectura
IX (intent exclusive, intencional exclusivo)	Espacios de tablas, tablas	Intención de actualizar filas
SIX (share with intent Exclusive, compartido intencional exclusivo)	Tablas	Sin bloqueos de lectura en las filas pero con bloqueos X en las filas actualizadas
U (Update, actualización)	Filas, tablas	Bloqueo de actualización pero permitiendo leer a otros
NX (next key exclusive, siguiente clave exclusivo)	Filas	Bloqueo de la siguiente clave para inserciones y borrados para prevenir las lecturas fantasma durante las exploraciones de índice RR
X (exclusive, exclusivo)	Filas, tablas	Sólo se permiten lectores no comprometidos
Z (superexclusive, superexclusivo)	Espacios de tablas, tablas	Acceso completo exclusivo

FIGURA 26.7. Modos de bloqueo de DB2.

cas para mayores detalles (en el Apartado 16.9 se describe una forma sencilla de bloqueo de la siguiente clave que elimina el problema fantasma).

La transacción puede establecer la granularidad del bloqueo en el nivel de tabla mediante el uso de una instrucción de bloqueo de tabla (una extensión SQL). Esto es útil para aplicaciones que conocen que su nivel deseado de aislamiento está en el nivel de tabla. También DB2 elige las granularidades de bloqueo apropiadas cuando se ejecutan utilidades tales como reorganización de la base de datos y bloqueo. Las versiones sin conexión de estas utilidades normalmente bloquean la tabla en modo exclusivo. Las versiones en conexión de las utilidades permiten que otras transacciones se ejecuten concurrentemente adquiriendo bloqueos de filas.

En cada base de datos se ejecuta un agente de detección de interbloqueos que periódicamente verifica los interbloqueos entre las transacciones. El intervalo para la detección de interbloqueos es un parámetro configurable. En caso de interbloqueo, el agente elige una víctima y la finaliza. La víctima produce un código de error SQL, indicando que la causa del fallo fue un interbloqueo.

26.5.2. Compromiso y retroceso

Las aplicaciones pueden comprometerse o retrocederse mediante el uso de las instrucciones explícitas **commit** y **rollback**. Las aplicaciones también pueden emitir instrucciones **begin transaction** y **end transaction** para controlar el ámbito de las transacciones. No se soportan las transacciones anidadas. Normalmente DB2 libera todos los bloqueos que se mantienen por una transacción en **commit** o **rollback**. Sin embargo, si se ha declarado una instrucción de cursor mediante la cláusula **withhold** entonces se mantienen algunos bloqueos durante los compromisos.

26.5.3. Registro histórico y recuperación

DB2 implementa el registro histórico y los esquemas de recuperación ARIES (el esquema ARIES se describe brevemente en el Apartado 17.9.6). Este registro histórico de escritura anticipada lo utiliza para enviar registros desde este registro histórico al archivo de registro histórico persistente, antes de que las páginas de datos se escriban en el compromiso. DB2 soporta dos tipos de modos de registro: registro histórico circular y registro de archivo. En el registro histórico circular, DB2 utiliza un conjunto predefinido de archivos de registro histórico primario y secundario. El registro histórico circular es útil para la recuperación de caídas o la recuperación de un fallo de la aplicación. En el registro histórico de archivo, DB2 crea nuevos archivos de registro histórico y guarda los archivos de registro histórico viejos con el fin de liberar espacio en el sistema de archivos. Los registros históricos de archivo son necesarios para la recuperación hacia adelante de una copia de seguridad de archivo (se describe con más detalle más adelante). En ambos casos DB2 permite al usuario configurar el número de archivos de registro histórico y los tamaños de los archivos de los registros históricos.

En entornos con grandes actualizaciones, DB2 puede configurarse para utilizar compromisos en grupo (Apartado 24.3) con el fin de combinar escrituras de registro histórico de varias transacciones y ejecutarlos utilizando una única operación E/S.

DB2 soporta *retroceso* de transacciones, recuperación de caídas, así como recuperaciones por instantes (point-in-time) o hacia adelante (roll-forward). En el

caso de una recuperación tras una caída, DB2 ejecuta las fases de *deshacer* estándar de procesamiento y procesamiento *rehacer* hasta y desde el último punto de revisión con el fin de recuperar el estado comprometido adecuado de la base de datos. Para la recuperación por instantes, DB2 puede restaurar la base de datos desde una copia de seguridad y avanzar a un punto específico en el tiempo utilizando los archivos históricos guardados. El comando de recuperación hacia adelante soporta tanto los niveles de bases de datos como de espacios de tablas. También se pueden emitir en nodos específicos sobre un sistema multinodo.

Recientemente se ha hecho disponible un esquema de recuperación en paralelo para mejorar el rendimiento en sistemas multiprocesador SMP mediante la utilización de muchas CPUs. DB2 ejecuta la recuperación coordinada a través de nodos MPP mediante un esquema global de puntos de revisión.

26.6. ARQUITECTURA DEL SISTEMA

La Figura 26.8 muestra algunos de los distintos procesos o hebras en un servidor DB2. Las aplicaciones remotas cliente se conectan al servidor de la base de datos a través de agentes de comunicación tales como *db2tcpcm*. Se asigna un agente a cada aplicación (agente coordinador en entornos MPP o SMP) denominado hebra *db2agent*. Este agente y sus agentes subordinados ejecutan las tareas relacionadas con la aplicación. Cada base de datos tiene un conjunto de procesos o hebras que ejecutan tareas tales como preextracción, limpieza de páginas de la cola de la memoria intermedia y detección de interbloqueos. Finalmente hay un conjunto de agentes en el entorno del servidor para ejecutar tareas tales como detección de caídas, creación de procesos, control de recursos del sistema y servicio de licencia. DB2 proporciona parámetros de configuración para controlar el número de hebras y procesos en un servidor. Casi todos los tipos distintos de agentes se pueden controlar mediante el uso de parámetros de configuración.

Modelo de procesamiento: partición única

FIGURA 26.8. Modelo de procesos en DB2.

La Figura 26.9 muestra los distintos tipos de segmentos de memoria en DB2. La memoria privada en los agentes o hebras se utiliza principalmente para variables locales y estructuras de datos que son relevantes solamente para la actividad actual. Por ejemplo, una ordenación privada podría asignar memoria desde el montículo privado del agente. La memoria compartida se divide en memoria compartida del servidor, memoria compartida de la base de datos y memoria compartida de la aplicación. El nivel de la base de datos de memoria compartida contiene estructuras de datos útiles tales como las colas de memoria intermedia, las listas de bloqueos, las cachés de los paquetes de aplicación y las áreas de ordenación compartida. Las áreas de memoria compartida del servidor y de la aplicación se utilizan principalmente para estructuras de datos comunes tales como parámetros de configuración del servidor y memorias intermedias de comunicaciones.

DB2 soporta varias colas de memoria intermedia para una base de datos. Las colas de memoria intermedia se pueden crear mediante el uso de la instrucción create bufferpool y se puede asociar con espacios de tablas. Es útil disponer de varias colas de memoria intermedia por diversas razones, pero se deberían definir después de un cuidadoso análisis de los requisitos de la carga de trabajo. DB2 también proporciona la capacidad de almacenamiento extendido para aprovechar memorias grandes (memorias mayores de 4 gigabytes) en sistemas que solamente tienen capacidad de direccionamiento de 32 bits. El almacenamiento extendido se utiliza como una memoria intermedia de copia de seguridad compartida para colas de memoria intermedia activas. Las páginas que se extraen o reemplazan de una cola de memoria intermedia activa se escriben en el área de almacenamiento extendido y se pueden volver a copiar desde allí si se necesita. Por ello, el almacenamiento extendido puede ayudar a evitar E/S en grandes sistemas de memoria.

DB2 soporta una completa lista de configuración de memoria y parámetros de ajuste. Esto incluye parámetros para todas las áreas de montículos de estructuras de datos grandes tales como las colas de memoria intermedia predeterminadas, el montículo de ordenación, la caché de paquetes, los montículos de control de la aplicación, el área de lista de bloqueos y cosas similares.

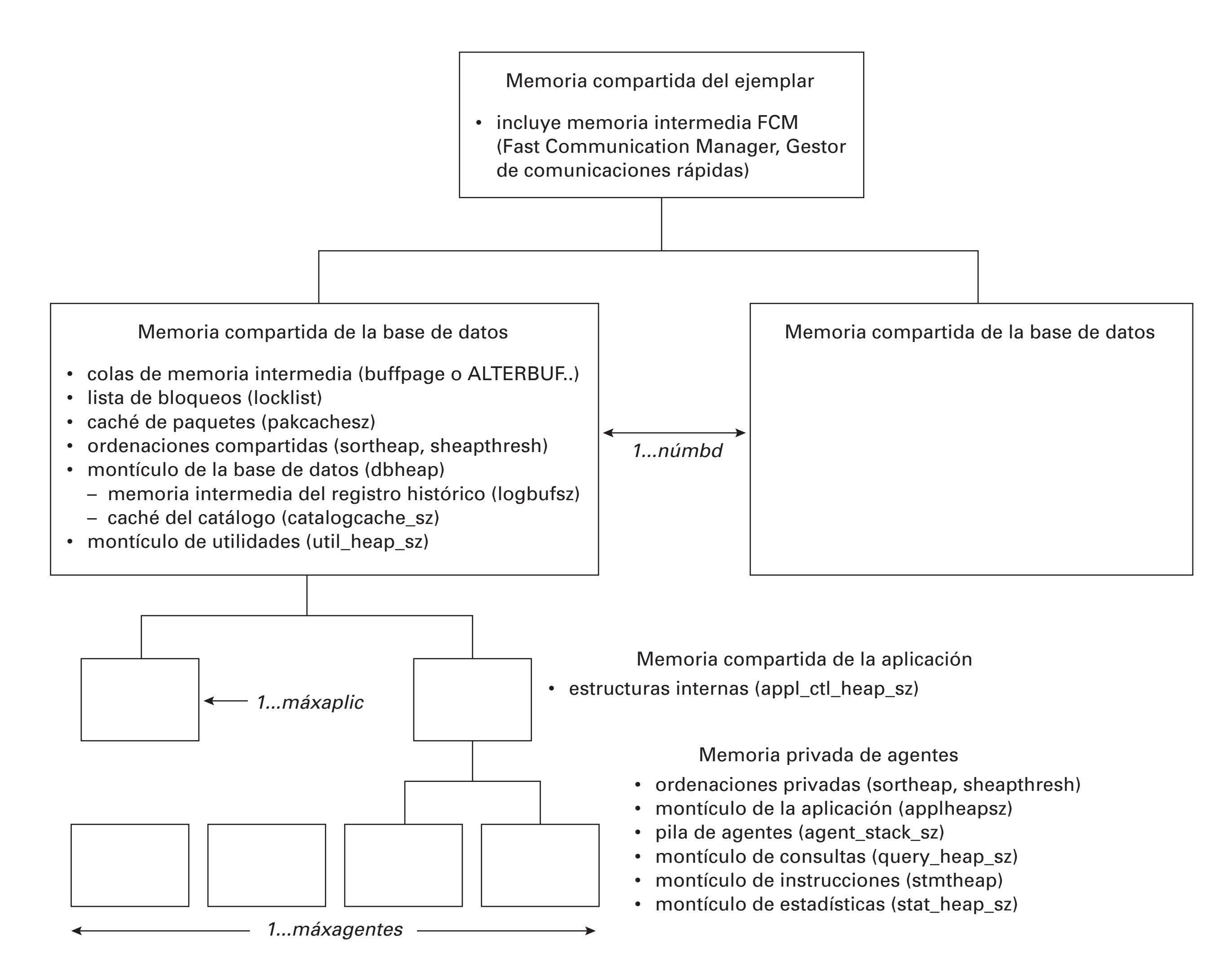


FIGURA 26.9. Modelo de memoria DB2.

26.7. RÉPLICAS, DISTRIBUCIÓN Y DATOS EXTERNOS

DB2 Data Propagator es un producto en la familia DB2 que proporciona réplica de datos entre DB2 y otros sistemas de bases de datos relacionales tales como Oracle, SQL Server de, SQL Server de Sybase e Informix, y orígenes de datos no relacionales tales como IMS de IBM.

Data Propagator consiste en componentes capturar y aplicar que se controlan mediante interfaces de administración. Los mecanismos de captura de cambios se basan en tablas DB2 basadas en registros históricos o basadas en disparadores en el caso de otros orígenes de datos. Esto es, para las tablas DB2 los cambios se detectan examinando el registro de la base de datos mientras que los disparadores se utilizan para detectar los cambios de otros orígenes de datos. Los cambios capturados se almacenan en áreas temporales de tablas bajo el control del propagador de datos DB2, que se aplican después a estas tablas intermedias, con cambios, a las tablas destino mediante el uso de instrucciones SQL (inserciones, actualizaciones y borrados). Las transformaciones basadas en SQL se pueden ejecutar sobre estas tablas intermedias utilizando condiciones de filtro además de agregaciones. Las filas resultantes se pueden aplicar a una o más tablas destino. Los medios de administración controlan todas estas acciones.

Otro miembro de la familia DB2 es el producto Data Joiner, que proporciona soporte a bases de datos federadas y distribuidas. Data Joiner puede integrar tablas en DB2 remotas u otras bases de datos relacionales en una única base de datos distribuida. Data Joiner proporciona un método basado en el coste para la optimización de consultas entre los distintos sitios de datos. Los datos no relacionales también se pueden integrar en Data Joiner mediante el uso de envolturas para crear datos tabulares.

DB2 soporta funciones de tabla definidas por el usuario que pueden permitir el acceso de orígenes de datos no relacionales y externos. DB2 crea funciones de tabla definidas por el usuario mediante la instrucción create function con la cláusula returns table. Con estas características DB2 puede participar en los protocolos OLE-DB.

Finalmente DB2 proporciona soporte completo para procesamiento de transacciones distribuidas mediante el protocolo de compromiso en dos fases. DB2 puede actuar como el coordinador o agente para el soporte de transacciones distribuidas. Como coordinador, DB2 puede ejecutar todos los estados del protocolo de compromiso en dos fases. Como participante, DB2 puede interactuar con cualquiera de los administradores de transacciones distribuidas comerciales.

26.8. HERRAMIENTAS DE ADMINISTRACIÓN DE BASES DE DATOS

DB2 proporciona una serie de herramientas para facilitar el uso y administración. Las herramientas creadas por los fabricantes han permitido la mejora del núcleo del conjunto de herramientas del programa.

El centro de control DB2 es la herramienta primaria para el uso y administración de bases de datos DB2. El centro de control se ejecuta sobre muchas plataformas del tipo estación de trabajo. Tiene interfaces para administrar objetos de distintos tipos, tales como servidores, bases de datos, tablas e índices. Además contiene interfaces orientadas en las tareas para ejecutar comandos y permite a los usuarios generar secuencias de comandos SQL.

La Figura 26.10 es una pantalla del panel principal del centro de control. Muestra una lista de tablas en la base de datos Sample en la instancia DB2 sobre el nodo CranKarm. El administrador puede utilizar el menú para invocar un conjunto de herramientas componentes. Los componentes principales del centro de control son el centro de órdenes, el centro de guiones, diario, gestión de licencias, centro de alertas, supervisor del rendimiento, explicación visual, administración de bases de datos remotas, gestión de almacenamiento y soporte para la réplica. El centro de órdenes permite a los usuarios y administradores enviar órdenes de la base de datos y ejecutar instrucciones SQL. El centro de guiones permite a los usuarios ejecutar guiones SQL construidos de forma interactiva o desde un archivo. El supervisor del rendimiento permite al usuario supervisar varios eventos en el sistema de la base de datos y obtener instantáneas del rendimiento. SmartGuides proporciona ayuda para la configuración de parámetros del sistema DB2. Un constructor de procedimientos almacenados ayuda al usuario a desarrollar e instalar estos procedimientos. La explicación visual da al usuario vistas gráficas del plan de ejecución de la consulta. Un asistente de índices ayuda al administrador sugiriendo índices de rendimiento.

Aunque el centro de control es una interfaz integrada de muchas de las tareas, DB2 también proporciona acceso directo a la mayoría de las tareas. Para los usuarios las herramientas tales como la característica de explicación, las tablas de explicación y la explicación gráfica proporcionan un análisis detallado de los planes de consulta. Los usuarios pueden utilizar el centro de control para modificar las estadísticas (si se permite la modificación) con el fin de generar los mejores planes de consulta.

Para los administradores, DB2 proporciona un soporte completo para la carga, importación, exportación, reorganización, redistribución y otras utilidades rela-



FIGURA 26.10. Centro de control DB2.

cionadas con los datos. Además, DB2 soporta herramientas tales como:

- Auditoría para el mantenimiento de la traza de auditoría de las acciones sobre la base de datos
- Regulador para controlar la prioridad y tiempos de ejecución en distintas aplicaciones
- Supervisor de consultas para gestionar los trabajos de consulta en el sistema
- Características de traza y diagnóstico para la depuración
- Supervisión de eventos para seguir los recursos y eventos durante la ejecución del sistema.

26.9. RESUMEN

Este capítulo proporciona una breve sinopsis de características disponibles en DB2. Como resumen, DB2 es un servidor de bases de datos multi-plataforma, dimensionable y relacional orientado a objetos. En las notas bibliográficas se citan libros y guías más completos sobre DB2.

NOTAS BIBLIOGRÁFICAS

Chamberlin [1996] y Chamberlin [1998] proporcionan una buena revisión de las características de SQL y programación de DB2. Los primeros libros de Date [1989] y Martin et al. [1989] proporcionan un buen repaso de las características de DB2 para OS/390. Los manuales de DB2 proporcionan revisiones definitivas de varias versiones de DB2. La mayoría de estos manuales están disponibles en línea en el sitio http://www.software.ibm.com/data/pubs. Libros recientes que proporcionan un entrenamiento práctico para el uso y administración de DB2 son Zikopoulos et al. [2000], Sanders [2000] y Cook et al. [1999]. Finalmente, Prentice Hall está publicando una serie completa de libros sobre el enriquecimiento y certificación sobre varios aspectos de DB2.

La investigación de IBM sobre XML incluye Shanmugasundaram et al. [2000] y Carey et al. [2000]. IBM ha sido un participante activo en la estandarización de XQuery. Se puede encontrar detalles sobre la extensión XML de DB2 en línea en el sitio

http://www.software.ibm.com/data/db2/extenders/xmlext.

Para una descripción detallada de los modos de bloqueo y su uso véase DB2 Administration Guide (capítulo Application Considerations), que está disponible en línea en http://www.software.ibm.com/data/pubs. Mohan [1990a] y Mohan y Levine [1992] proporcionan descripciones detalladas del control de concurrencia de índices en DB2.

Las contribuciones a la investigación de IBM, algunas de las cuales se listan más adelante, han mejorado continuamente el producto DB2. Chamberlin et al. [1981] proporciona una historia y evaluación del proyecto System R, que jugó una función crucial en el desa-

rrollo de bases de datos relacionales y condujo al desarrollo del producto DB2. El problema Halloween y otros aspectos de la historia de System R aparecen en

http://www.mcjones.org/System R/SQL Reunion 95/index.html.

Las técnicas de procesamiento de transacciones tales como el registro histórico de escritura anticipada y los algoritmos de recuperación ARIES se describen en Mohan et al. [1992]. El procesamiento y optimización de consultas en Starbrust se describen en Haas et al. [1990]. El procesamiento en paralelo en DB2 Parallel Edition se describe en Baru et al. [1995]. El soporte de bases de datos activas, incluyendo las restricciones y los disparadores, se describen en Cochrane et al. [1996]. Carey et al. [1999] describe el soporte relacional orientado a objetos en DB2.

CAPÍTULO

27

SQL SERVER DE MICROSOFT

Sameet Agarwal, José A. Blakeley, Thomas Casey, Kalen Delaney, César Galindo-Legaria, Goetz Graefe Michael Rys, Michael Zwilling
Microsoft

SQL Server de Microsoft es un sistema gestor de bases de datos relacionales que se usa desde en portátiles y ordenadores de sobremesa hasta en servidores corporativos, con una versión compatible, basada en el sistema operativo PocketPC, disponible para dispositivos de bolsillo, tales como PocketPCs y lectores de código de barras. SQL Server se desarrolló originalmente en los años 80 en SyBase para sistemas UNIX y posteriormente pasado a sistemas Windows NT para Microsoft. Desde 1994 Microsoft ha lanzado versiones de SQL Server desarrolladas independientemente de Sybase, que dejó de utilizar el nombre SQL Server a finales de los años 90. La última versión disponible es SQL Server 2000, disponible en ediciones personales, para desarrolladores, estándar y corporativa, y traducida a muchos lenguajes en todo el mundo. En este capítulo el término SQL Server se refiere a todas estas ediciones de SQL Server 2000.

SQL Server proporciona servicios de réplica entre varias copias de SQL Server así como con otros sistemas de bases de datos. Sus Analysis Services (servicios de análisis), una parte integral del sistema, incluye dispositivos de procesamiento en conexión analítico (OLAP, Online Analytical Processing) y recopilación de datos. SQL Server proporciona una gran colección de herramientas gráficas y «asistentes» que guían a los administradores de las bases de datos por tareas tales como establecer copias de seguridad regulares, réplica de datos entre servidores y ajuste del rendimiento de una base de datos. Muchos entornos de desarrollo soportan SQL Server, incluyendo Visual Studio de Microsoft y productos relacionados, en particular los productos y servicios .NET.

27.1. HERRAMIENTAS PARA EL DISEÑO Y CONSULTA DE BASES DE DATOS

SQL Server proporciona un conjunto de herramientas para gestionar todos los aspectos del desarrollo de SQL Server, consulta, ajuste, verificación y administración. La mayoría de estas herramientas se centran alrededor del Administrador corporativo de SQL Server. El administrador corporativo es un complemento accesorio de Microsoft Management Console (MMC), una herramienta que proporciona una interfaz común para trabajar con varias aplicaciones del servidor en una red Windows.

27.1.1. Desarrollo de bases de datos y herramientas visuales

Mientras se diseña una base de datos, el administrador de la base de datos crea objetos de bases de datos tales como tablas, coulmnas, claves, índices, relaciones, restricciones y vistas. Para ayudar a crear estos objetos el Administrador corporativo de SQL Server proporciona acceso a herramientas visuales de bases de datos. Estas herramientas proporcionan tres mecanismos para ayudar al diseño de la base de datos: el diseñador de bases de datos, el diseñador de tablas y el diseñador de vistas. Las herramientas visuales también proporcionan una herramienta de consulta visual que permite al administrador de la base de datos el uso de capacidades de arrastrar y soltar para construir consultas visualmente.

El diseñador de bases de datos es una herramienta visual que permite al administrador de la base de datos crear tablas, columnas, claves, índices, relaciones y restricciones. Con el diseñador de bases de datos, un usuario puede interactuar con los objetos de la base de datos mediante diagramas de base de datos, los cuales muestran de forma gráfica la estructura de la base de datos. El usuario puede crear y modificar objetos que son visibles sobre diagramas (tablas, columnas, relaciones y claves) y algunos objetos que no son visibles en los diagramas (índices y restricciones). La Figura 27.1 muestra el diagrama de una base de datos abierto con el Administrador corporativo.

27.1.2. Herramientas de consulta y ajuste de las bases de datos

SQL Server proporciona herramientas para ayudar al proceso de desarrollo de aplicaciones. Se pueden desarrollar y verificar inicialmente las consultas y procedimientos almacenados utilizando el Analizador de con-

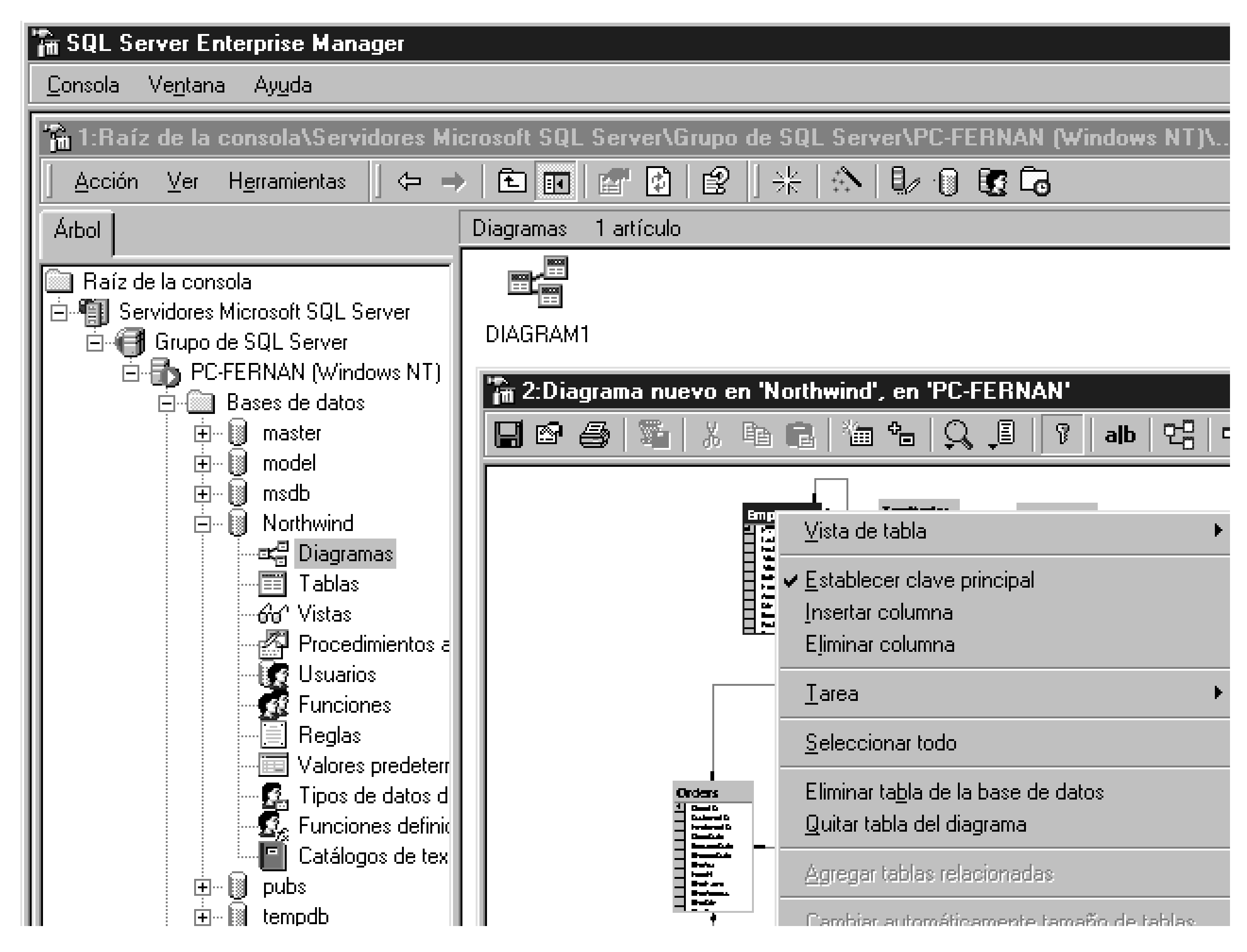


FIGURA 27.1. Diagrama que muestra las opciones del diseñador de tablas para la tabla de empleados.

sultas de SQL Server, el cual proporciona herramientas básicas de consulta y ajuste de las bases de datos. Se pueden realizar otros análisis utilizando el Analizador de SQL Server. Las recomendaciones del ajuste de índices vienen proporcionadas por una tercera herramienta, el Asistente de optimización de índices.

27.1.2.1. Analizador de consultas de SQL

EL Analizador de consultas de SQL proporciona una interfaz de usuario sencilla y gráfica para ejecutar consultas SQL y ver los resultados. Permite varias ventanas de forma que pueden existir conexiones de bases de datos simultáneas (una o más instalaciones de SQL Server). Un desarrollador de consultas puede elegir tener resultados de consulta mostrados en una ventana de texto o en una rejilla. El Analizador de consultas de SQL proporciona una representación gráfica de **showplan**, los pasos elegidos por el optimizador para la ejecución de la consulta. También proporciona informes opcionales de los comandos reales procesados por SQL Server (una traza en el cliente) y el trabajo realizado por el cliente. El Analizador de consultas de SQL viene con un explorador de objetos y permite al usuario arrastrar y soltar objetos o nombres de tablas en la tabla. También incluye la herramienta de depuración de Transact-SQL. El depurador permite a un usuario depurar paso a paso cualquier procedimiento almacenado, examinando el comportamiento de los parámetros, las variables locales y las funciones del sistema mientras se ejecutan estos pasos.

Un administrador o desarrollador de la base de datos puede utilizar el Analizador de consultas de SQL para:

- Analizar consultas: el Analizador de consultas de SQL Server puede mostrar un plan de ejecución gráfico o contextual para cualquier plan de consultas, así como mostrar estadísticas relacionadas con el tiempo y recursos requeridos para ejecutar cualquier plan.
- Dar formato a las consultas SQL: el Analizador de consultas permite la sangría y su eliminación en las líneas de código, cambio de la caja de las palabras o secciones de código, comentar una única o varias líneas y mostrar las consultas con un código de color controlado por el usuario.

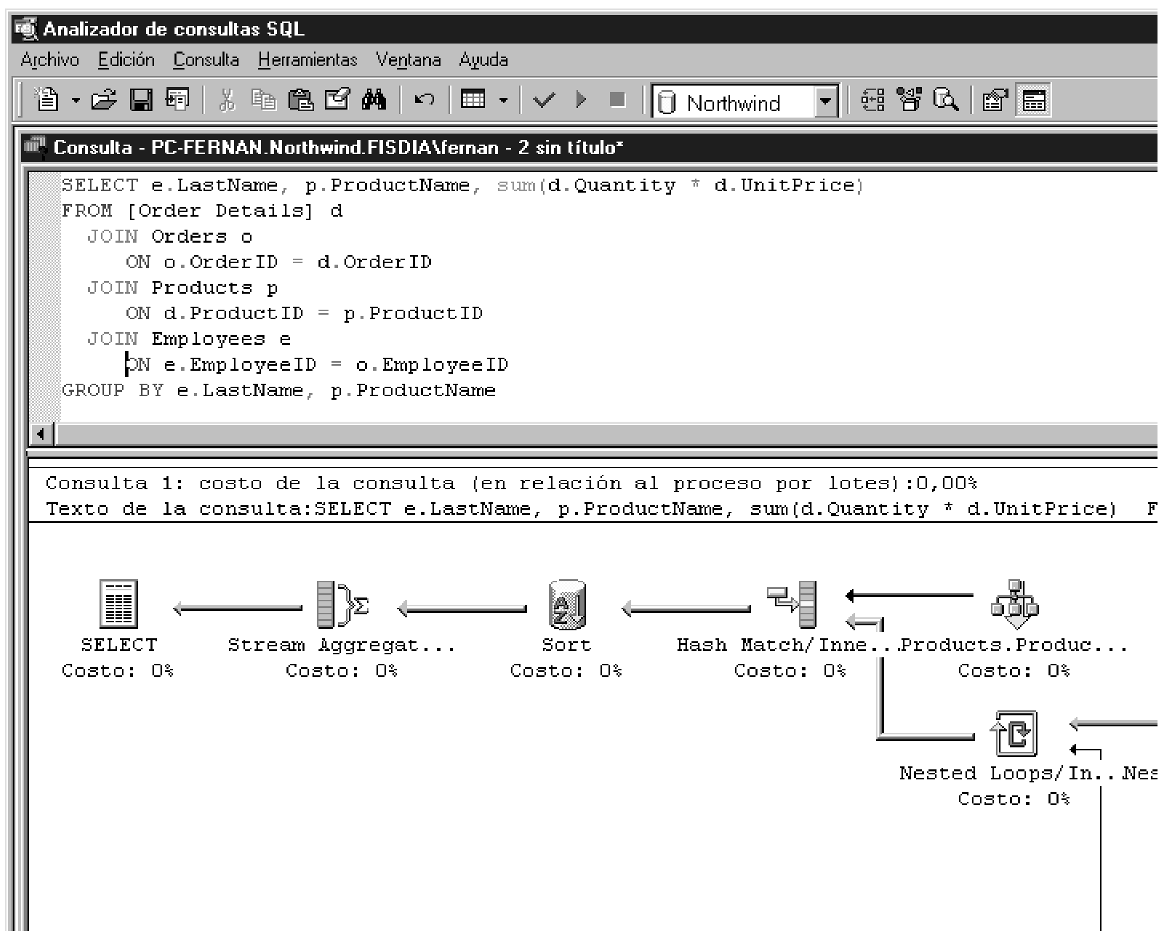


FIGURA 27.2. Un plan de ejecución con showplan para una reunión de cuatro tablas con una agregación group by.

- Utilizar plantillas para procedimientos almacenados, funciones e instrucciones SQL básicas: el Analizador de consultas viene con docenas de plantillas predefinidas para construir instrucciones LDD, y los usuarios pueden definir las suyas propias. Cuando se ejecuta una plantilla los usuarios pueden proporcionar valores específicos para los nombres de objetos y columnas, tipos de datos y otra información específica.
- Arrastrar nombres de objetos desde el Explorador de objetos a la ventana Consulta: el Analizador de consultas permite al desarrollador elegir la definición de un objeto o, para tablas y vistas, ver plantillas para crear instrucciones **insert, update, delete** o **select**.
- Definir teclas de acceso directo y opciones de la barra de herramientas personales: el Analizador de consultas permite definir teclas de acceso directo para una ejecución rápida de consultas comunes y proporciona un control completo sobre los comandos que están disponibles como botones en la tabla de herramientas y en qué posición aparecen los botones.

La Figura 27.2 muestra el Analizador de consultas mostrando el plan de ejecución gráfico para una consulta que involucra una reunión de cuatro tablas y una agregación.

27.1.2.2. Analizador de SQL

El Analizador de SQL es una utilidad gráfica que permite a los administradores de la base de datos supervisar y registrar la actividad de la misma. El Analizador de SQL puede mostrar toda la actividad del servidor en tiempo real o puede crear filtros que se centren en las acciones de usuarios particulares, aplicaciones o tipos de órdenes. El Analizador de SQL puede mostrar cualquier instrucción SQL o procedimiento almacenado enviado a cualquier ejemplar de SQL Server (si los privilegios de seguridad lo permiten) así como los datos de rendimiento que indican cuánto tiempo la consulta tardó en ejecutarse y cuánta CPU y E/S fue necesaria y el plan de ejecución que utilizó la consulta.

El Analizador de SQL permite ahondar aún más en SQL Server para supervisar automáticamente toda instrucción ejecutada como parte de un procedimiento almacenado, toda operación de modificación de datos, todo bloqueo adquirido o liberado, o cada vez que cre-

ce un archivo de base de datos. Hay docenas de eventos distintos que se pueden capturar y docenas de elementos de datos que se pueden capturar para cada evento. SQL Server realmente divide la funcionalidad de traza en dos componentes separados aunque conectados. El Analizador de SQL está en la traza del cliente. Mediante el uso del Analizador de SQL un usuario puede elegir guardar los datos capturados a un archivo o una tabla, además de mostrarlos en la interfaz de usuario del Analizador (IU). El Analizador muestra todo evento que cumple el criterio del filtro mientras ocurre. Una vez que se han guardado los datos de la traza, el Analizador de SQL puede leer los datos guardados para mostrarlos o analizarlos.

En el lado del servidor está la traza de SQL, que gestiona las colas de eventos generados por productores de eventos. Una hebra consumidora lee los eventos desde las colas y los filtra antes de enviarlos al proceso que las solicitó. Los eventos son la unidad principal de actividad en lo que se refiere a la traza, y un evento puede ser cualquier cosa que suceda dentro de SQL Server o entre SQL Server y un cliente. Por ejemplo, la creación o eliminación de un objeto, la ejecución de un procedimiento almacenado, la adquisición o liberación de un bloqueo, y el envío de un archivo de proceso por lotes Transact-SQL desde el cliente a SQL Server son eventos. Hay un conjunto de procedimientos almacenados del sistema para definir qué eventos se deberían seguir, qué datos son interesantes para cada evento o dónde guardar la información recogida por los eventos. Los filtros aplicados a los eventos pueden reducir la cantidad de información recogida y almacenada. La Figura 27.3 muestra la ficha de eventos desde el cuadro de diálogo de definición de traza del Analizador de SQL.

SQL Server proporciona más de 100 eventos predefinidos y un usuario puede configurar 10 eventos adicionales. Las trazas en el lado del servidor se ejecutan de una forma invisible y se puede iniciar una traza de una forma automática cada vez que se inicia SQL Server. Esto garantiza que siempre se recogerá cierta información crítica y se puede utilizar como un mecanismo útil de auditoria. SQL Server está certificado para un nivel C2 de seguridad y muchos de los eventos que pueden ser objeto de traza están disponibles exclusivamente para soportar requisitos C2 de certificación.

27.1.2.3. Asistente para la optimización de índices

Con todas las posibles técnicas de procesamiento de la consulta disponibles el optimizador de consultas puede determinar un plan de consulta razonablemente efectivo incluso en ausencia de índices bien planeados. Sin embargo, esto no significa que una base de datos bien ajustada no se beneficie de buenos índices. El diseño de los mejores índices posibles para las tablas en una base de datos grande es una tarea compleja, no solamente requiere un conocimiento completo de cómo SQL Server utiliza los índices y cómo el optimizador de consultas realiza sus decisiones sino cómo se utilizan realmente los datos por las aplicaciones y consultas interactivas. El Asistente para optimización de índices de SQL Server es una herramienta poderosa para el diseño de los mejores índices basados en la consulta observada y cargas de actualización.

El asistente ajusta una única base de datos cada vez y fundamenta sus recomendaciones sobre una carga de trabajo que puede ser un archivo de eventos de traza capturados o un archivo con instrucciones SQL. El Analizador de SQL está diseñado para capturar todas las instrucciones SQL enviadas por todos los usuarios en un cierto periodo de tiempo. El asistente puede entonces examinar los patrones de acceso a los datos para todos los usuarios, aplicaciones y tablas, y realizar recomendaciones razonables.

27.1.3. Administrador corporativo de SQL Server

Además de proporcionar acceso a las herramientas de diseño y visuales de bases de datos, el Administrador corporativo de SQL, de fácil uso, soporta administración centralizada de todos los aspectos de varias instalaciones de SQL Server, incluyendo la seguridad, eventos, alertas, programación, copias de seguridad, configuración del servidor, ajuste, búsqueda de texto completo y réplicas. EL Administrador corporativo de SQL Server permite a un administrador de la base de datos crear, modificar y copiar esquemas y objetos de la base de datos SQL Server tales como tablas, vistas y desencadenadores. Debido a que se pueden organizar en grupos varias instalaciones de SQL Server y ser tratadas como una unidad, el Administrador corporativo de SQL Server puede gestionar cientos de servidores simultáneamente.

Aunque se puede ejecutar en la misma computadora que el motor de SQL Server, el Administrador corporativo de SQL Server ofrece las mismas capacidades de gestión cuando se ejecuta en cualquier máquina basada en Windows NT/2000. El Administrador corporativo de SQL Server también se ejecuta sobre Windows 98 aunque no están disponibles algunas capacidades en este entorno (es de mención la capacidad de utilizar el Administrador de control de servicios, una característica de Windows NT/2000, para iniciar y parar SQL Server de forma remota). Además, la arquitectura eficiente cliente/servidor de SQL Server hace práctico el uso de capacidades de acceso remoto (acceso telefónico a redes) de Windows NT/2000 así como Windows 98 para la administración y gestión.

El Administrador corporativo de SQL Server evita que el Administrador de la base de datos tenga que conocer los pasos específicos y la sintaxis para completar un trabajo. Proporciona más de 20 asistentes para guiar al administrador de la base de datos en el proceso de configurar y mantener una instalación de SQL Server. La interfaz del Administrador corporativo aparece en la Figura 27.4.

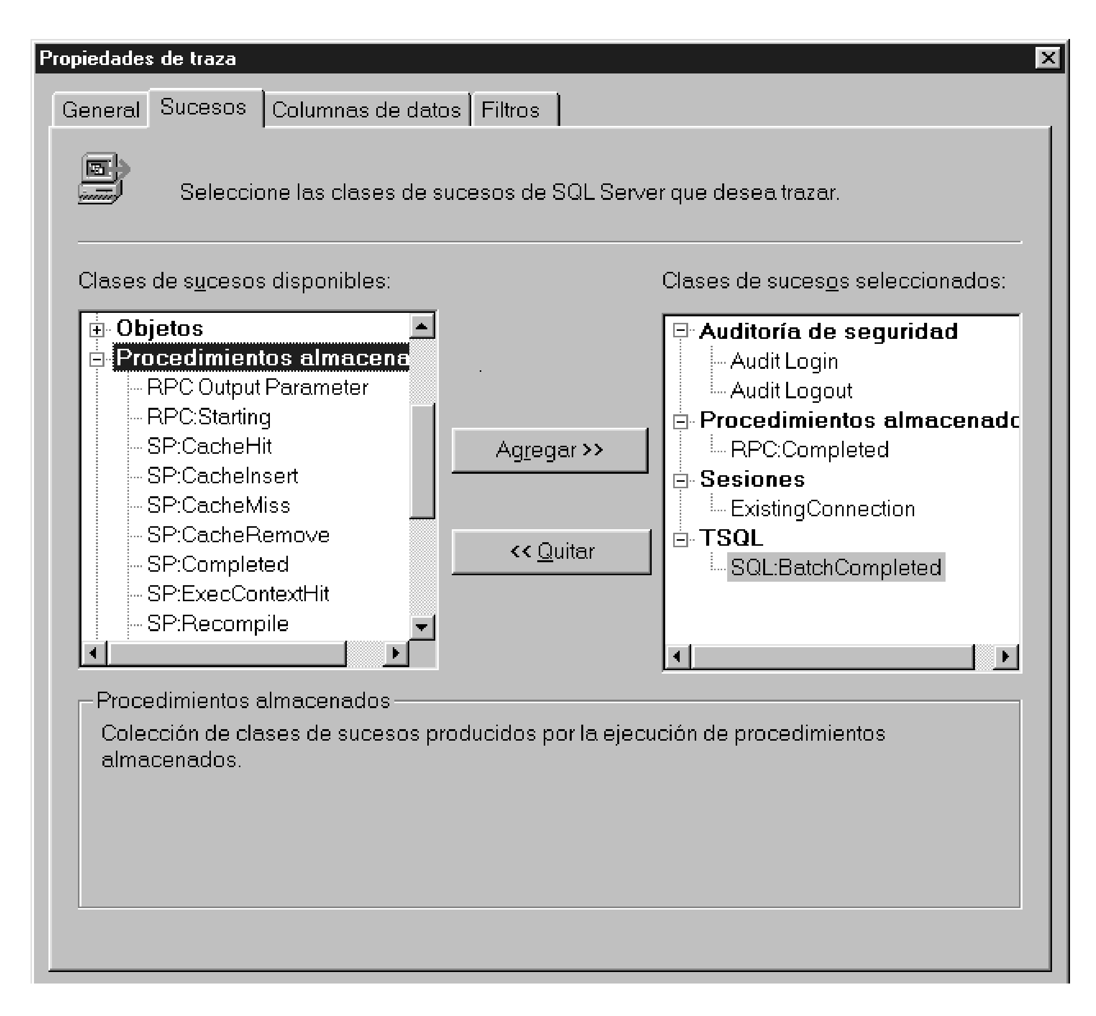


FIGURA 27.3. Ficha Eventos del cuadro de diálogo Propiedades de traza.

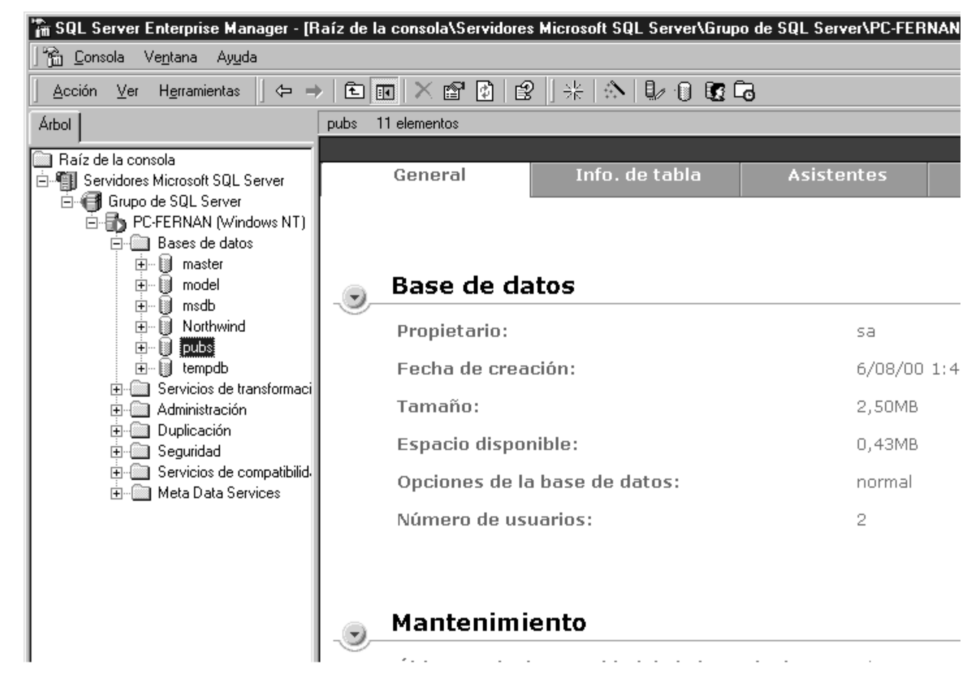


FIGURA 27.4. Interfaz del Administrador corporativo de SQL Server.

27.2. VARIACIONES Y EXTENSIONES DE SQL

Transact-SQL es el lenguaje de bases de datos soportado por SQL Server. Transact-SQL es un lenguaje completo de programación de bases de datos que incluye instrucciones de definición y manipulación de datos, instrucciones iterativas y condicionales, variables, procedimientos y funciones. Transact-SQL cumple el nivel de entrada de la norma SQL-92 pero también soporta varias características desde los niveles intermedios y superiores. Transact-SQL también soporta extensiones a la norma SQL-92.

27.2.1. Tipos de datos

SQL Server proporciona un conjunto de tipos de datos primitivos que definen todos los tipos de datos que se pueden utilizar con SQL Server. El conjunto de tipos de datos primitivos incluyen:

- Un conjunto completo de tipos de enteros con signo con 1, 2, 4 y 8 bytes de precisión (tinyint, smallint, int y bigint)
- Un tipo de datos bit con valores 0 o 1 (bit)
- Tipo decimal con 38 dígitos de precisión (decimal, numeric)
- Tipos de moneda con precisión de 1/1000 de la unidad monetaria (money, smallmoney)
- Tipos de fecha y hora con una precisión de hasta 3.33 milisegundos (datetime, small-datetime)
- Tipos en coma flotante de precisión sencilla y doble (real, float)
- Tipos de cadenas de caracteres de tamaño fijo y variable de hasta 2^{30}-1 caracteres Unicode y no Unicode (char/nchar, varchar/nvarchar, text/ntext)
- Cadenas de bytes de tamaño fijo y variable de hasta 2^{31}-1 bytes (binary, varbinary, image)
- Un tipo cursor que permite referencias a un objeto cursor (cursor)
- Tipos de datos de identificadores únicos globales y para la base de datos

Además, SQL Server soporta los tipos sql_variant y table descritos en los apartados 27.2.1.1 y 27.2.1.2.

27.2.1.1. Tipo Variant

Sql_variant es un tipo de datos escalar que permite a una columna de una fila, variable o argumento de función contener valores de cualquier tipo escalar SQL (excepto text, ntext, image, rowversion y sql_variant). Es utilizado por aplicaciones que necesitan almacenar datos cuyo tipo no se puede conocer cuando se definen los datos. Internamente el sistema guarda el tipo original de datos. Es posible filtrar, reunir y ordenar columnas sql_variant. La función del sistema sql_variant_property puede devolver detalles sobre los datos reales almacenados en una columna de tipo sql_variant, incluyendo el tipo base e información del tamaño.

27.2.1.2. Tipo Table

Table es un tipo que permite a una variable guardar un conjunto de filas. Este tipo se utiliza principalmente para especificar el tipo devuelto por las funciones que devuelven tabla. Una variable table se comporta como una variable local. Tiene un ámbito bien definido, que es la función, procedimiento almacenado, o proceso por lotes en el cual se declara. Dentro de este ámbito se puede utilizar una variable table como una tabla normal. Se puede aplicar en cualquier lugar donde se utiliza una tabla o expresión de tabla en instrucciones **select, insert, update y delete**.

Las variables table se limpian automáticamente al final de la función, procedimiento almacenado o proceso por lotes en el cual se definen. Además, las variables table utilizadas en procedimientos almacenados resultan en menos recompilaciones de estos procedimientos que cuando se utilizan las tablas temporales. Las transacciones que involucran las variables table duran solamente en la actualización de la variable table por lo que las variables table requieren menos recursos de bloqueos y de registro histórico.

27.2.2. Funciones definidas por el usuario

Las funciones definidas por el usuario permite a los mismos definir sus propias funciones Transact-SQL mediante el uso de la instrucción **create function**. SQL Server soporta las funciones que devuelven un tipo escalar o una tabla. Las funciones escalares se pueden utilizar en cualquier expresión escalar dentro de una instrucción LMD o LDD de SQL. Las funciones que devuelven tablas se pueden utilizar en cualquier lugar donde se permita una tabla en una instrucción **select**. Las funciones que devuelven una tabla cuyo cuerpo contiene una única instrucción SQL **select** se tratan como una vista (entre líneas expandida) en la consulta que hace referencia a la función. Puesto que las funciones que devuelven una tabla permiten la introducción de argumentos, las funciones entre líneas que devuelven tablas se pueden considerar vistas parametrizadas.

27.2.3. Vistas

Una *vista* es una tabla virtual cuyos contenidos están definidos por una instrucción **select**. Las vistas son un poderoso mecanismo de modelado de datos y seguridad. Las vistas indexadas (Apartado 27.2.3.1) también pueden proporcionar un beneficio sustancial en el ren-

dimiento. Las tablas referenciadas por la definición de la vista se conocen como tablas base. En el ejemplo que sigue, *vistatítulos* es una vista que selecciona los datos desde tres tablas base: *título*, *autortítulo* y *títulos*. Estas tablas son parte de la base de datos ejemplo *pubs* incluida con SQL Server.

```
create view vistatítulos as
    select título, au_ord, au_nombre, precio, ventas,
    id_editorial
    from autores as a join autortítulo as at on (a.au_id
    = at.au_id)
        join títulos as t on (t.título_id = at.título_id)
```

Se puede hacer referencia a la vista *vistatítulos* en instrucciones de la misma forma a como se haría con una tabla base:

```
select *
from vistatítulos
where precio >= 30
```

Una vista puede hacer referencia a otras vistas. Por ejemplo *vistatítulos* presenta información que es útil para administradores, pero las empresas normalmente revelan sus datos de venta sólo en los informes trimestrales o anuales. Se puede construir una vista que seleccione todas las columnas de vistatítulos excepto au_ord y ventas. Esta nueva vista puede ser utilizada por los clientes para obtener listas de los libros disponibles sin ver la información financiera:

```
create view Cust vistatítulos as
select título, au_nombre, precio, id_editorial
from vistatítulos
```

Las consultas que emplean las vistas se optimizan expandiendo la definición de la vista en la consulta (en otras palabras, el optimizador basado en el coste optimiza toda la consulta como si se hubiera realizado sin el uso de vistas).

27.2.3.1. Vistas indexadas

Además de las vistas tradicionales como se definieron en la norma ANSI de SQL, SQL Server soporta las vistas indexadas (materializadas). Las vistas indexadas pueden mejorar sustancialmente el rendimiento de las consultas complejas de ayuda a la toma de decisiones que recuperan un gran número de filas y agregan grandes cantidades de información en sumas recuentos y medias. SQL Server soporta la creación de índices agrupados en una vista y subsecuentemente cualquier número de índices no agrupados. Una vez que se indexa una vista, el optimizador puede utilizar sus índices en consultas que hacen referencia a la vista o sus tablas base. Las consultas existentes se pueden beneficiar de la eficiencia mejorada de la recuperación de los datos directamente de la vista indexada sin tener que ser reescrita para hacer referencia a la vista. Las instrucciones de actualización de las tablas base se propagan automáticamente a los índices de la vista.

27.2.3.2. Vistas divididas

Las vistas divididas se utilizan para dividir los datos entre varias tablas, bases de datos o ejemplares de SQL Server con el fin de distribuir la carga de trabajo. Si los datos se dividen entre varios servidores, se pueden proporcionar los mismos beneficios en el rendimiento que un agrupamiento de servidores y se puede utilizar para soportar las necesidades de procesamiento de los mayores sitios Web o centros de datos corporativos. Una tabla base se puede dividir en varias tablas miembro, cada una de las cuales tiene un subconjunto de filas de la tabla original. Cada servidor debe tener definición de la tabla dividida. Una vista dividida utiliza el operador **union** para combinar todas las divisiones en las tablas base en un único conjunto de resultados que se comporta exactamente como una copia de la tabla original completa. Por ejemplo, una tabla de clientes se puede dividir entre tres servidores. Cada servidor debe tener una división de la tabla base disjunta como sigue:

- En Servidor1:
```
    create table Cliente_33 (
        IDCliente integer primary key
            check (IDCliente between 1 and
            32999),
    ... Otras definiciones de columna)
```
- En Servidor2:
```
    create table Cliente_66 (
        IDCliente integer primary key
            check (IDCliente between 33000 and
            65999),
        ... Otras definiciones de columna)
```
- En Servidor3:
```
    create table Cliente_99 (
        IDCliente integer primary key
            check (IDCliente between 66000 and
            99999),
        ... Otras definiciones de columna)
```

Después de crear las tablas miembro, el administrador de la base de datos define una vista dividida distribuida en cada servidor, cada vista teniendo el mismo nombre. La vista dividida distribuida para Servidor1 se definiría como sigue:

```
create view Clientes as
select * from BaseDeDatos.PropietarioTabla.
Clientes_33
union all
select * from Server2.BaseDeDatos.Propietario
Tabla.Clientes_66
union all
select * from Server3.BaseDeDatos.Propietario
Tabla.Clientes_99
```

Sobre Server2 y Server3 se definirían vistas similares.

El hecho de tener la vista definida sobre cada servidor permite a las consultas que hacen referencia al nombre de la vista dividida y distribuida ejecutarse en cada uno de los servidores miembro. El sistema opera como si en cada servidor miembro hubiera una copia completa de la tabla original, aunque cada servidor tenga solamente una tabla miembro local y una vista dividida y distribuida que referencia a la tabla local y las dos tablas remotas. La ubicación de los datos es transparente a la aplicación. Con estas tres vistas, cualquier instrucción Transact-SQL sobre cualquiera de los tres servidores que haga referencia a Clientes verá los mismos resultados que desde la tabla original. Se puede acceder a las vistas divididas no solamente mediante instrucciones **select** sino también mediante instrucciones **insert**, **update** y **delete**, incluyendo instrucciones que afectan a varias divisiones e incluso aquellas que requieren trasladar filas de una división a otra.

27.2.3.3. Vistas actualizables

Generalmente las vistas puede ser el objetivo de las instrucciones **update, delete** o **insert** si la modificación de los datos se aplica a solamente una de las tablas base de la vista. Las actualizaciones de las vistas divididas se pueden propagar a varias tablas base. Por ejemplo, la siguiente instrucción **update** incrementará los precios para el editor «0736» en un 10 por ciento.

```
update vistatítulos
set precio = precio * 1.10
where id_editorial = ’0736’
```

Para las modificaciones de los datos que afectan a más de una tabla base, la vista se puede actualizar si hay un desencadenador **instead** (Apartado 27.2.4) definido para la operación. Los desencadenadores **insert** para las operaciones **insert, update** o **delete** se pueden definir en una vista para especificar las actualizaciones que se deben ejecutar en las tablas base para implementar las modificaciones correspondientes en la vista.

27.2.4. Desencadenadores (disparadores)

Los desencadenadores son procedimientos Transact-SQL que se ejecutan automáticamente cuando se envía una instrucción **update, insert** o **delete** a una tabla base o vista. Los desencadenadores son un mecanismo que posibilita la aplicación de la lógica del negocio de forma automática cuando se modifican los datos. Los desencadenadores pueden extender la lógica de verificación de la integridad de restricciones declarativas, predeterminadas y reglas, aunque las restricciones declarativas se deberían utilizar preferentemente siempre que se satisfagan las necesidades.

Hay dos clases generales de desencadenadores que difieren en el tiempo con respecto a la instrucción de desencadenamiento, bajo la que se realiza la acción. Los desencadenadores **after** se ejecutan después de la instrucción de desencadenamiento y se aplican posteriores restricciones declarativas. Los desencadenadores **instead** se ejecutan en lugar de la acción de desencadenamiento. Los desencadenadores **instead** son similares a los desencadenadores **before**, pero realmente reemplazan la acción de desencadenamiento. En SQL Server los desencadenadores **after** se pueden definir solamente sobre tablas base mientras que los desencadenadores **instead** se pueden definir sobre tablas base o vistas. Los desencadenadores **instead** permiten que se pueda actualizar prácticamente cualquier vista.

27.3. ALMACENAMIENTO E INDEXACIÓN

En SQL Server una base de datos se refiere a una colección de archivos que contienen datos y que son soportados por un único registro histórico de transacciones. La base de datos es la unidad principal de administración en SQL Server y también proporciona un contenedor para estructuras físicas tales como tablas e índices y para estructuras lógicas tales como restricciones, vistas, etc.

27.3.1. Grupos de archivos

Con el fin de gestionar el espacio en una base de datos de forma efectiva, el conjunto de archivos en una base de datos se divide en grupos denominados grupos de archivos (filegroups). Cada grupo de archivos contiene uno o más archivos del sistema operativo.

Toda base de datos tiene al menos un grupo de archivos conocido como el grupo de archivos primario. Este grupo de archivos contiene todos los metadatos de la base de datos en tablas del sistema. El grupo de archivos primario también puede contener datos de usuario.

Si se crean grupos de archivos definidos por el usuario adicionales, un usuario puede controlar de forma explícita la ubicación de las tablas individuales, índices o las columnas de objetos grandes de una tabla ubicándolas en un grupo de archivos. Por ejemplo, el usuario puede elegir almacenar una tabla en el grupodearchivosA, su índice no agrupado en grupodearchivosB y las columnas de objetos grandes de la tabla en grupodearchivosC. La ubicación de estas tablas e índices en distintos grupos de archivos permite al usuario controlar el uso de los recursos de hardware (esto es, discos y el subsistema E/S). Un grupo de archivos en concreto se

considera siempre el grupo de archivos predeterminado; inicialmente, el grupo de archivos predeterminado es el grupo de archivos primario, pero se puede dar a cualquier grupo de archivos definido por el usuario la propiedad de determinado. Si una tabla o índice no está ubicada específicamente en un grupo de archivos, se crea en el grupo de archivos predeterminado.

27.3.2. Administración del espacio en grupos de archivos

Uno de los muchos propósitos principales es permitir una gestión de espacio efectiva. Todos los archivos de datos se dividen en unidades de 8 Kbytes de espacio fijo denominadas páginas. El sistema de asignación es responsable de asignar estas páginas a tablas e índices. El objetivo del sistema de asignación es minimizar la cantidad de espacio malgastado mientras que, al mismo tiempo, mantener el nivel de fragmentación de la base de datos en el mínimo para asegurar un buen rendimiento de exploración. Con el fin de lograr este objetivo el administrador de asignación normalmente asigna y desasigna todas las páginas en unidades de ocho páginas contiguas denominadas extensiones.

El sistema de asignación gestiona estas extensiones mediante varios mapas de bits. Estos mapas de bits permiten al sistema de asignación encontrar una página o extensión para la asignación de una forma rápida. Estos mapas de bits también se utilizan cuando se ejecuta una tabla completa o exploración de índices. La ventaja de usar mapas de bits basados en la asignación para la exploración es que permiten recorridos en el orden del disco de todas las extensiones que pertenecen a una tabla o en el nivel de las hojas de los índices, lo que mejora significativamente el rendimiento de la exploración.

Si hay más de un archivo en un grupo de archivos, el sistema de asignación proporciona extensiones para cualquier objeto en ese grupo de archivos utilizando un algoritmo de «relleno proporcional». Cada archivo se rellena con la proporción de la cantidad de espacio libre en ese archivo comparado con otros archivos. Esto llena todos los archivos de un grupo de archivos aproximadamente con el mismo factor y permite al sistema utilizar todos los archivos en un grupo de archivos de forma equitativa. Durante una exploración que utiliza los mapas de bits de asignación, el algoritmo de exploración aprovecha los distintos archivos enviando E/S separadas a cada uno de los archivos.

Una de las mayores decisiones al configurar una base de datos es determinar su tamaño. SQL Server permite a los archivos de datos cambiar su tamaño después de crear la base de datos. El usuario puede incluso elegir hacer que el archivo de datos crezca automáticamente si la base de datos se queda sin espacio. Por ello, el usuario puede configurar la base de datos a una aproximación razonable del tamaño esperado, pero hace que los archivos de la base de datos crezcan y se ajusten al patrón de uso, si la aproximación inicial no es correcta. SQL Server permite a los archivos disminuir. Con el fin de disminuir un archivo de datos, SQL Server traslada todos los datos desde el fin físico del archivo a un punto más cercano al inicio del archivo y entonces realmente reduce el archivo, devolviendo el espacio al sistema operativo.

27.3.3. Tablas

SQL Server soporta dos organizaciones diferentes para las tablas: montículos e índices agrupados. En una tabla organizada en montículos la ubicación de cada fila de la tabla se determina completamente mediante el sistema y no es especificada de ninguna forma por el usuario. Las filas de un montículo tienen un identificador fijo como identificador de fila (RID, Row Identifier) y su valor nunca cambia a no ser que se reduzca el archivo y la fila se traslade. Si la fila se torna lo suficientemente grande para no caber en la página en la que se insertó originalmente, el registro se mueve a un lugar distinto, pero se deja un resguardo en la ubicación original de forma que el registro todavía se puede encontrar utilizando su RID original.

En una organización agrupada por índices para una tabla, las filas de la tabla se almacenan en un árbol B+ ordenado mediante la clave de agrupamiento del índice. La clave de índice agrupado también sirve como el identificador único para cada fila. La clave para un índice agrupado se puede definir como no único, en cuyo caso SQL Server agrega una columna oculta adicional para hacer la clave única. El índice agrupado también sirve como una estructura de búsqueda para identificar una fila de la tabla en una clave particular o explorar un conjunto de filas de la tabla con claves con un cierto rango.

27.3.4. Índices

La forma más común de indexación son los índices no agrupados, que también se conocen como índices secundarios. Estos índices son árboles B+ sobre una o más columnas de la tabla base. Permiten acceso eficiente de una fila de una tabla base basada en un criterio de búsqueda sobre las columnas indexadas. Además, las consultas que se refieren solamente a las columnas que están disponibles mediante índices secundarios se procesan mediante la recuperación de las páginas desde el nivel hoja de los índices sin tener que recuperar los datos del índice agrupado o montículo.

SQL Server soporta la adición de columnas calculadas a una tabla. Una columna calculada es una columna cuyo valor es una expresión, normalmente basada en el valor de otras columnas en esa fila. SQL Server permite al usuario construir índices secundarios sobre columnas calculadas.

27.3.5. Exploraciones y lectura anticipada

La ejecución de las consultas en SQL Server pueden involucrar una serie de distintos modos de exploración

de las tablas e índices subyacentes. Éstos incluyen exploraciones ordenadas y desordenadas, exploraciones en serie y paralelas, unidireccionales y bidireccionales, hacia delante y hacia atrás, y exploración de toda la tabla o índice y exploraciones de rango o filtradas.

Cada uno de los modos de exploración tiene un mecanismo de lectura anticipada que intenta anticiparse a las necesidades del plan de ejecución con el fin de reducir la latencia y gastos y utilizar el tiempo sin trabajo del disco. El algoritmo de lectura anticipada de SQL Server utiliza el conocimiento del plan de ejecución de la consulta con el fin de conducir la lectura anticipada y asegurarse de que solamente se leen los datos que son realmente necesarios. Además, la cantidad de lectura anticipada se dimensiona de forma automática según el tamaño de la memoria intermedia, la cantidad de E/S que el subsistema del disco puede sostener y la velocidad a la que el plan de ejecución puede consumir los datos.

27.4. PROCESAMIENTO Y OPTIMIZACIÓN DE CONSULTAS

El procesador de consultas de SQL Server está basado en un entorno extensible que permite una rápida incorporación de nuevas técnicas de ejecución y optimización. La ejecución encapsula los algoritmos de procesamiento de datos en *iteradores* que se comunican entre sí utilizando la interfaz GetNextRow(). La optimización genera alternativas utilizando transformaciones en árbol y estima el coste de ejecución utilizando modelos detallados del comportamiento de las selectividades e iteradores.

27.4.1. Visión general de los procesos de optimización

Las consultas complejas presentan oportunidades significativas de optimización que requieren la ordenación de los bloques de consulta, con selección del plan basado en el coste estimado. SQL Server utiliza un entorno puramente algebraico. El entorno de optimización de SQL Server se basa en el prototipo de optimizador Cascades. Una instrucción SQL se compila como sigue:

- **Análisis/vinculación.** El optimizador analiza la instrucción y resuelve los nombres de tablas y columnas mediante el uso de catálogos. Resuelve e incorpora vistas para generar un árbol de operadores. Verifica la caché del procedimiento para ver si ya hay un plan para la consulta, en cuyo caso se evita la optimización. El árbol de operadores utiliza un álgebra relacional extendida donde no hay noción de bloque de consulta o tabla derivada sino simplemente una combinación arbitraria de operadores relacionales.
- **Simplificación/normalización.** El optimizador aplica reglas de simplificación sobre el árbol de operadores para obtener un formulario normal y simplificado. Estas reglas implementan transformaciones tales como enviar las selecciones hacia abajo y la simplificación de reuniones externas en reuniones. Durante la simplificación, el optimizador determina y carga las estadísticas requeridas para la estimación de la cardinalidad. Si se pierden las estadísticas requeridas, el optimizador las crea automáticamente antes de continuar la optimización.
- **Optimización basada en el coste.** El optimizador aplica la exploración e implementación de reglas para generar alternativas, estimar el coste de la ejecución y elegir el plan con el coste anticipado más bajo. Las reglas de exploración implementan la reordenación de operadores, incluyendo reordenación de la reunión y de la agregación. Las reglas de implementación introducen alternativas en la ejecución tales como reuniones por mezcla y reuniones por asociación.
- **Preparación del plan.** El optimizador crea estructuras del plan de ejecución para el plan seleccionado.

La optimización basada en el coste no se divide en fases que optimizan distintos aspectos de la consulta de forma independiente y no está restringida a una única dimensión tal como la enumeración de reuniones. En su lugar una colección de reglas de transformación define el espacio de interés y la estimación del coste se utiliza uniformemente para seleccionar un plan eficiente.

27.4.2. Simplificación de la consulta

Durante la simplificación, el optimizador envía las selecciones del árbol de operadores tan abajo como sea posible. Verifica los predicados en busca de contradicciones teniendo en cuenta las restricciones declaradas. Utiliza las contradicciones para identificar subexpresiones vacías, que se eliminan del árbol. Un escenario común es la eliminación de ramas **union** que recuperan los datos de las tablas con distintas restricciones.

Una serie de reglas de simplificación son *dependientes del contexto*, es decir, la sustitución solamente es válida en el contexto de la utilización de la subexpresión. Por ejemplo, una reunión externa se puede simplificar en una reunión interna si una operación de filtrado posterior elimina reglas no coincidentes que se rellenaron con **null**. Otro ejemplo es la eliminación de reuniones sobre claves externas que no se necesitan ejecutar si no hay uso posterior de las columnas desde la tabla referenciada. Un tercer ejemplo es el contexto de independencia de duplicados, que especifica que la dis-

tribución de una o más copias de una fila no afecta al resultado de la consulta. Las subexpresiones bajo semirreuniones y bajo **distinct** son independientes de duplicados, lo que permite cambiar **union** a **union all**.

Para agrupar una agregación se utiliza el operador *GbAgg*, que crea grupos y opcionalmente aplica una función de agregado sobre cada grupo. La eliminación de duplicados, expresado en SQL mediante la palabra clave **distinct** es sencillamente un *GbAgg* sin funciones de agregado a calcular. Durante la simplificación, la información sobre las claves y dependencias funcionales se utiliza para reducir el agrupamiento de columnas.

Las subconsultas se normalizan eliminando especificaciones de consulta correlacionadas y utilizando algunas variantes de la reunión en su lugar. La eliminación de correlaciones no es una «estrategia de ejecución de subconsultas» sino simplemente un paso de normalización. Se consideran entonces una serie de estrategias de ejecución consideradas durante la optimización basada en el coste.

27.4.3. Reordenación y optimización basada en el coste

En SQL Server las transformaciones se integran completamente en la generación basada en el coste y selección de planes de ejecución. Además de la reordenación de la reunión interna, el optimizador de consultas emplea transformaciones de reordenación para los operadores reunión externa, semirreunión y antisemirreunión del álgebra relacional estándar (con duplicados, para SQL). *GbAgg* se reordena también, trasladándolo debajo de las reuniones siempre que sea posible. La agregación parcial, esto es, la introducción de un nuevo *GbAgg* con agrupación sobre un superconjunto de las columnas de un *GbAgg* que se encuentre más arriba, se considera debajo de las reuniones y **union all** y también en planes paralelos. Véanse las referencias dadas en las notas bibliográficas para más detalles.

La ejecución correlacionada se considera durante la exploración del plan; el caso más simple es una reunión de búsqueda en el índice. SQL Server modela esto utilizando *Apply*, que opera sobre una tabla T y una expresión relacional parametrizada $E(t)$. *Apply* ejecuta E para cada fila de T, que proporciona los valores de los parámetros. La ejecución correlacionada se considera como una alternativa a la ejecución, sin considerar el uso de las subconsultas en la formulación SQL original. Es una estrategia muy eficiente cuando la tabla T es muy pequeña y los índices soportan la ejecución parametrizada eficiente de $E(t)$. Además se considera la reducción del número de ejecuciones de $E(t)$ donde hay valores de parámetros duplicados mediante dos técnicas: ordenar T según los valores de los parámetros de forma que se reutilice un único resultado de $E(t)$ mientras que el valor del parámetro no cambia, o también utilizar una tabla de asociación que siga la pista del resultado de $E(t)$ para (algún subconjunto) de valores anteriores del parámetro.

Algunas aplicaciones seleccionan filas según el resultado de algún agregado para su grupo. Por ejemplo «Hallar los clientes cuyo saldo es mayor que el doble de la media para su segmento de mercado». La formulación SQL requiere una autorreunión. Durante la exploración se detecta este patrón y se considera la ejecución por segmentos como una alternativa a la autorreunión. La utilización de la vista materializada también se considera durante la optimización basada en el coste. El encaje de vistas interactúa con la ordenación de operadores en la que el uso puede que no sea aparente hasta que se realice otra reordenación. Cuando se encuentra que una vista se ajusta a alguna subexpresión, la tabla que contiene el resultado de la vista se agrega como alternativa a la expresión correspondiente. Dependiendo de la distribución de datos e índices disponibles puede ser mejor o no que la expresión original (la selección se realizará basándose en la estimación del coste).

Para estimar el coste de ejecución del plan, el modelo tiene en cuenta el número de filas que se esperan procesar, denominado el objetivo filas, así como el número de veces que se ejecuta una subexpresión. El objetivo filas puede ser menor que la estimación de la cardinalidad en casos tales como *Apply*/*semijoin*. *Apply*/*semijoin* devuelve la fila t de T tan pronto como $E(t)$ produce una única fila (es decir, comprueba $E(t)$). Por tanto, el objetivo filas de la salida de $E(t)$ es 1, y los objetivos filas de los subárboles de $E(t)$ se calculan para $E(t)$ para este objetivo y se usan para la estimación del coste.

27.4.4. Planes de actualización

Los planes de actualización optimizan el mantenimiento de índices, verifican las restricciones, aplican acciones en cascada y mantienen las vistas materializadas. Para el mantenimiento de los índices, en lugar de tomar cada fila y mantener todos sus índices los planes de actualización aplican modificaciones por índice, ordenamiento de filas y aplican la operación de actualización según el orden de la clave. Esto minimiza las operaciones E/S aleatorias, especialmente cuando el número de filas a optimizar es grande. Las restricciones se manejan con un operador **assert**, que ejecutan un predicado y envían un error si el resultado es **false**. Las restricciones de integridad referencial se definen mediante predicados **exist**, los cuales se convierten en semirreuniones y se optimizan considerando todos los algoritmos de ejecución.

El problema Halloween se soluciona utilizando elecciones basadas en el coste. El problema Halloween se refiere a la siguiente anormalidad: supongamos que se lee en orden ascendente un índice sueldo y los salarios se suben un 10 por ciento. Como resultado de la actualización, las filas se moverán hacia arriba en el índice y se volverán a encontrar y actualizar de nuevo, llevándonos a un bucle infinito. Una forma de solventar este problema es separar el procesamiento de dos fases: en primer lugar se leen todas las filas que se actualizarán y se hace una copia de ellas en algún lugar temporal, después se

leen de este lugar y se aplican todas las actualizaciones. Otra alternativa es leer desde un índice distinto donde las filas no se trasladarán como resultado de la actualización. Algunos planes de ejecución proporcionan la separación de las fases de forma automática, si se ordena o construye una tabla de asociación en las filas a actualizar. En el optimizador de SQL Server la protección Halloween se modela como una propiedad de los planes. Se generan varios planes que proporcionan la propiedad requerida y se selecciona uno según el coste de ejecución estimado.

27.4.5. Búsqueda parcial y heurísticas

Los optimizadores basados en el coste se enfrentan al problema de la explosión del espacio de búsqueda puesto que las aplicaciones emiten consultas que involucran docenas de tablas. Para solucionar esto, SQL Server utiliza varios estados de optimización, cada uno de los cuales utiliza transformaciones de la consulta para explorar regiones sucesivamente mayores del espacio de búsqueda.

Hay transformaciones simples y completas diseñadas para la optimización exhaustiva, así como transformaciones inteligentes que implementan varias heurísticas. Las transformaciones inteligentes generan planes que están muy lejos en el espacio de búsqueda, mientras que las transformaciones sencillas exploran las cercanías. Los estados de optimización aplican una mezcla de ambas clases de optimización, en primer lugar enfatizando en las transformaciones inteligentes y posteriormente cambiando a transformaciones sencillas. Se preservan los resultados óptimos en los subárboles, de forma que los estados posteriores se pueden beneficiar de los resultados generados con anterioridad. Cada estado necesita equilibrar técnicas de generación de planes opuestas:

- **Generación exhaustiva de alternativas:** para generar el espacio completo el optimizador debería utilizar transformaciones completas, locales, no redundantes (una regla de transformación equivalente a una secuencia de más transformaciones primitivas solamente introduce costes adicionales).
- **Generación heurística de candidatos:** una serie de candidatos interesantes (seleccionados según el coste estimado) probablemente están lejos en términos de reglas de transformación primitivas. Aquí las transformaciones deseadas son incompletas, globales y redundantes.

La optimización se puede terminar en cualquier punto después de que el primer plan se haya generado. Tal terminación se basa en el coste estimado del mejor plan encontrado y el tiempo gastado ya en la optimización. Por ejemplo, si una consulta requiere solamente mirar unas pocas filas en algunos índices, se producirá rápidamente un plan muy barato en los primeros estados, terminando la optimización. Este enfoque nos ha permitido agregar nuevas heurísticas fácilmente a lo largo del tiempo, sin comprometer la selección basada en el coste de los planes o la exploración exhaustiva del espacio de búsqueda, cuando es apropiado.

27.4.6. Ejecución de la consulta

Los algoritmos de ejecución soportan el procesamiento basado en la ordenación y basado en la asociación, y sus estructuras de datos se diseñan para optimizar el uso de la caché del procesador. Las operaciones de asociación soportan agregación y reunión básica, con una serie de optimizaciones, extensiones y ajuste dinámico del sesgo de datos. La operación *flow-distinct* es una variante de asociación con valores distintos (hash distinct), donde las filas se devuelven tan pronto como se encuentra un nuevo valor distinto, en lugar de esperar a procesar toda la entrada. Este operador es efectivo para consultas que utilizan **distinct** y solicita solamente unas pocas filas, como con la constructora **top n**. Los planes correlacionados la ejecución de $E(t)$, a menudo incluyendo varias búsquedas en el índice basadas en el parámetro para cada fila t de la tabla T. La *preextracción asíncrona* permite la emisión de varias solicitudes de búsqueda en el índice al motor de almacenamiento. Se implementa de la siguiente forma: se hace una solicitud de búsqueda en el índice sin bloqueo para una fila t de T, entonces t se sitúa en una cola de preextracción. Se sacan las filas de la cola y son utilizadas por *Apply* para ejecutar $E(t)$. La ejecución de $E(t)$ no requiere que los datos ya estén listos en la memoria intermedia, pero tener buenas operaciones de preextracción maximiza la utilización del hardware e incrementa el rendimiento. El tamaño de la cola se determina dinámicamente como una función de aciertos en caché. Si no se requiere ninguna ordenación de las filas de salida de *Apply*, las filas de esta cola se pueden descartar para minimizar la espera en la E/S.

La ejecución en paralelo se implementa mediante el operador *Exchange*, que gestiona varias hebras, particiones o datos de difusión y alimenta los datos a varios procesos. El optimizador de consultas decide la ubicación de *Exchange* según el coste estimado. El grado de paralelismo se determina dinámicamente en tiempo de ejecución, según la utilización actual del sistema.

Los planes de índices están formados de los trozos descritos anteriormente. Por ejemplo, se considera el uso de una reunión de índices para resolver las conjunciones de predicados (o unión de índices para las disyunciones), basándose en el coste. Dicha reunión se puede realizar en paralelo, utilizando cualquiera de los algoritmos de reunión del servidor. También se consideran reuniones de índices para el único propósito de ensamblar una fila con el conjunto de columnas necesario en una consulta, que es algunas veces más rápido que explorar una tabla base. Tomar identificadores de registros de un índice secundario y localizar la fila correspondiente de la tabla base es efectivamente equivalente a ejecutar una reunión de búsqueda en índice.

Para ello se usan las técnicas genéricas de ejecución correlacionada como la preextracción asíncrona.

La comunicación con el motor de almacenamiento se realiza mediante OLE-DB, lo que permite acceder a otros proveedores de datos que implementan esta interfaz. OLE-DB es un mecanismo utilizado para consultas distribuidas y remotas, que las maneja directamente el procesador de consultas. Los proveedores de datos se clasifican según el rango de funcionalidad que proporcionan, desde simples proveedores de conjuntos de filas sin capacidades de indexación a proveedores con soporte completo de SQL.

27.5. CONCURRENCIA Y RECUPERACIÓN

Los subsistemas de transacciones, registro histórico, bloqueos y recuperación aseguran las propiedades ACID esperadas de un sistema de bases de datos.

27.5.1. Transacciones

En SQL Server todas las instrucciones son atómicas y las aplicaciones pueden especificar varios niveles de aislamiento para cada instrucción. Las transacciones se utilizan para encuadrar una secuencia de instrucciones, haciendo el conjunto completo atómico y controlando su aislamiento desde otras transacciones. Una única transacción puede incluir instrucciones que no solamente seleccionan, insertan, borrar o actualizan registros, sino que también crean o eliminan tablas, construyen índices y realizan importaciones masivas de datos. Las transacciones pueden abarcar bases de datos en servidores remotos. Cuando las transacciones se extienden por varios servidores, SQL Server utiliza un servicio del sistema operativo Windows, denominado coordinador de transacciones distribuidas (Microsoft Distributed Transaction Coordinator, MS DTC) para ejecutar un procesamiento de compromiso de dos fases. MS DTC soporta el protocolo de transacción XA y, junto con OLE-DB, proporciona el fundamento para transacciones ACID entre sistemas heterogéneos.

27.5.1.1. Puntos de almacenamiento

SQL Server soporta dos tipos de puntos de almacenamiento: de instrucciones y con nombre. Los puntos de almacenamiento de instrucciones son puntos tomados al comienzo de una instrucción de forma que, si una instrucción falla, se puede retroceder sin tener que retroceder toda la transacción. Un punto de almacenamiento con nombre es una instrucción **save transaction** enviada por una aplicación que etiqueta un punto en la transacción. Se puede enviar una instrucción **rollback** posterior para retroceder hasta el punto con nombre. Un retroceso a un punto de almacenamiento no libera bloqueos y no se puede utilizar en transacciones distribuidas.

27.5.1.2. Opciones de concurrencia para actualizaciones

SQL Server ofrece control de concurrencia optimista y pesimista para las operaciones de actualización.

El control de concurrencia optimista funciona bajo la suposición de que los conflictos de recursos entre varios usuarios son poco probables (aunque no imposibles) y permite a las transacciones ejecutarse sin bloquear ningún recurso. Solamente cuando se intentan cambiar los datos se verifican los recursos para determinar si han ocurrido conflictos. Si sucede un conflicto, la aplicación debe leer los datos e intentar el cambio de nuevo. Las aplicaciones pueden elegir si se detectan los cambios comparando los valores o verificando la columna especial rowversion de una fila. El control de concurrencia optimista requiere el uso de cursores.

El control de concurrencia pesimista bloquea los recursos cuando se requieren durante la duración de una transacción. A no ser que ocurran interbloqueos se asegura la finalización satisfactoria de una transacción. El control de concurrencia pesimista es el predeterminado para SQL Server.

Recurso	*Descripción*
RID	Identificador de fila, usado para bloquear una única fila de una tabla
Clave	Bloqueo de fila en un índice; protege los rangos de clave en transacciones secuenciables
Página	Página de tabla o índice de 8 Kbyte
Extensión	Grupo contiguo de ocho páginas de datos o de índice
Tabla	Tabla completa, incluyendo todos los datos e índices
BD	Base de datos

FIGURA 27.5. Recursos bloqueables.

27.5.1.3. Niveles de aislamiento

SQL-92 define los siguientes niveles de aislamiento, todos ellos soportados por SQL Server:

- Lectura no comprometida (nivel inferior donde las transacciones se aíslan solamente para asegurar que no se leen físicamente datos corruptos).
- Lectura comprometida (Nivel predeterminado de SQL Server)
- Lectura repetible
- Secuenciable (nivel superior, donde las transacciones están completamente aisladas entre sí)

27.5.2. Bloqueos

SQL Server proporciona bloqueos de varias granularidades que permiten que una transacción bloquee distintos recursos (véase la Figura 27.5, donde los recursos se listan en orden creciente de granularidad). Para minimizar el coste del bloqueo, SQL Server bloquea los recursos automáticamente a una granularidad apropiada para la tarea. El bloqueo a una granularidad menor, tal como filas, aumenta la concurrencia, pero tiene un coste mayor, puesto que se deben realizar más bloqueos si se bloquean muchas filas. El bloqueo a una granularidad mayor, tal como tablas, es costoso en términos de concurrencia puesto que el bloqueo de una tabla completa restringe a otras transacciones el bloqueo a cualquier parte de la tabla, pero tiene unos costes de CPU y memoria menores, ya que se adquieren menos bloqueos.

Los modos de bloqueos disponibles son compartido (S, shared), de actualización (U, update) y exclusivo (X, exclusive). Los bloqueos de actualización se utilizan para evitar que ocurra una forma común de interbloqueo cuando varias sesiones están leyendo, bloqueando y potencialmente actualizando recursos más tarde. Otros modos de bloqueo adicionales (denominados bloqueos de rango de clave) se adoptan solamente en el nivel de aislamiento secuenciable para bloquear el rango entre dos filas y un índice.

Los bloqueos de varias granularidades requieren que se adquieran los bloqueos en una jerarquía estricta de mayor a menor granularidad. La jerarquía es la base de datos, tabla, página y fila. Cuando se intenta bloquear en un nivel superior al compartido, de actualización y exclusivo se usan bloqueos intencionales.

27.5.2.1. Bloqueo dinámico

El bloqueo de granularidad fina puede mejorar la concurrencia con el coste de ciclos de CPU y memoria extra para adquirir y mantener muchos bloqueos. Para muchas consultas una granularidad de bloqueo más burda proporciona mejor rendimiento sin pérdida de concurrencia (o mínima). Los sistemas de base de datos han requerido tradicionalmente sugerencias de consulta y opciones de tabla a las aplicaciones para especificar la granularidad del bloqueo. Además, hay parámetros de configuración (frecuentemente estáticos) para gestionar cuánta memoria dedicar a la administración del bloqueo.

En SQL Server la granularidad del bloqueo se optimiza automáticamente para un rendimiento y concurrencia óptimos para cada índice de una consulta. Además, la memoria dedicada al administrador de bloqueos se ajusta dinámicamente según la realimentación desde otras partes del sistema, incluyendo otras aplicaciones de la máquina.

La granularidad del bloqueo se optimiza antes de la ejecución de la consulta para cada tabla e índice utilizado en la consulta. El proceso de optimización del bloqueo tiene en cuenta el nivel de aislamiento (esto es, cuánto tiempo se mantienen los bloqueos), el tipo de exploración (rango, prueba o toda la tabla), el número estimado de filas a explorar, la selectividad (porcentaje de filas visitadas que son resultado de la consulta), densidad de filas (número de filas por página), tipo de operación (exploración, actualización), límites del usuario sobre la granularidad y memoria del sistema disponible.

La Figura 27.6 muestra una consulta ejemplo donde las filas resultado se identifican mediante una exploración de rango de un índice y después se recuperan las filas desde la tabla base. Aquí se utilizan los bloqueos de página sobre el índice, puesto que con un rango denso de filas del índice se requieren solamente unos pocos bloqueos de página. Sin embargo, las filas de la tabla base están dispersas por toda la tabla y por ello el bloqueo en un nivel de fila proporciona una concurrencia mucho mayor. En general, la optimización del bloqueo favorece la concurrencia en sus decisiones. Una vez se ejecuta una consulta, la granularidad de bloqueo se dimensiona automáticamente hasta el nivel de tabla si el sistema adquiere significativamente más bloqueos que los esperados por el optimizador o si la cantidad de memoria disponible cae y no se pueden soportar el número de bloqueos requeridos.

27.5.2.2. Detección de interbloqueos

SQL Server detecta de forma automática los interbloqueos que involucran bloqueos y otros recursos. Por ejemplo si la transacción A está manteniendo un bloqueo en Tabla1 y está esperando memoria disponible y la transacción B tiene algo de memoria que no puede compartir hasta que adquiera un bloqueo sobre Tabla1, la transacción presentará un interbloqueo. Las hebras y las memorias intermedias de comunicación también pueden estar involucradas en los interbloqueos. Cuando SQL Server detecta un interbloqueo, elige como la víctima del interbloqueo la transacción que es menos costosa de retroceder, considerando la cantidad de trabajo que la transacción ya ha realizado.

Frecuentemente la detección puede perjudicar al rendimiento del sistema. SQL Server automáticamente ajusta la frecuencia de la detección de interbloqueos a la frecuencia a la que están ocurriendo los interbloqueos. Si los interbloqueos no son frecuentes, el algoritmo de detección se ejecuta cada 5 segundos. Si son frecuen-

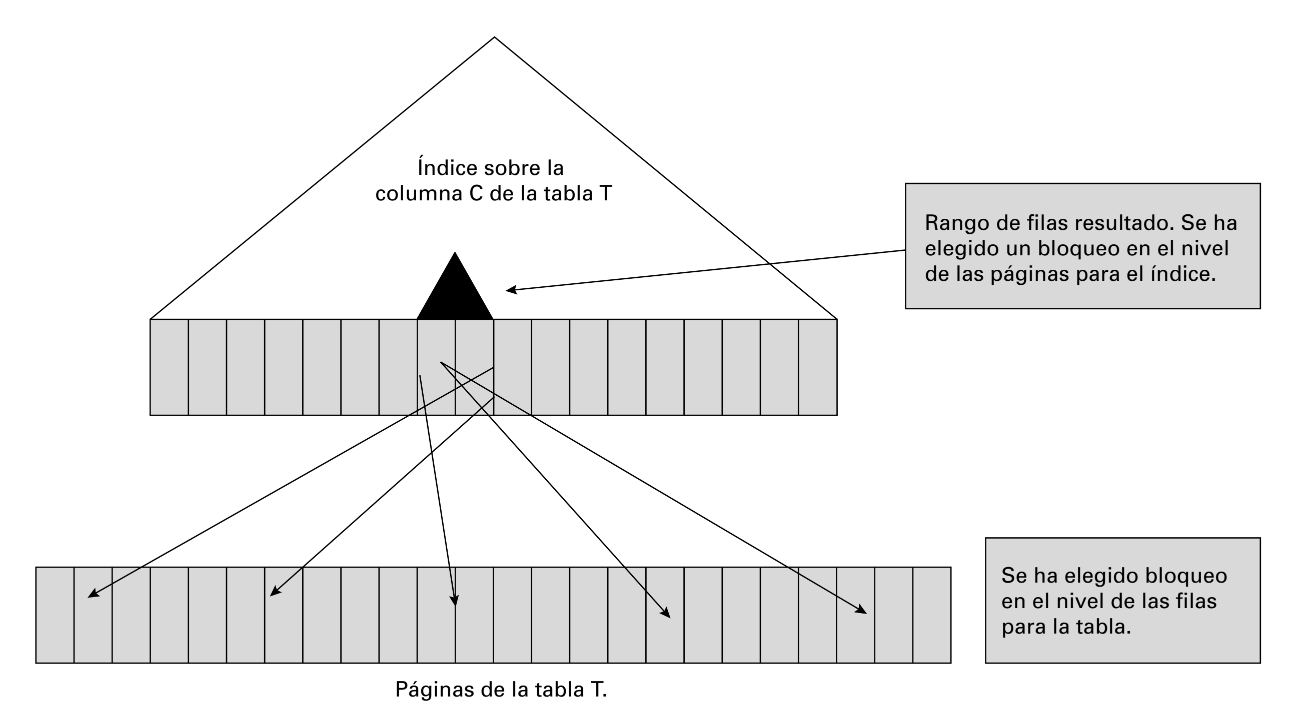


FIGURA 27.6. Granularidad de los bloqueos.

tes se comenzará a verificar cada vez que una transacción espera un bloqueo.

27.5.3. Registros históricos y recuperación

SQL Server está diseñado para recuperarse de fallos del sistema y de los medios, y el sistema de recuperación se puede dimensionar a máquinas con memorias intermedias muy grandes (100 Gbytes) y miles de unidades de disco.

27.5.3.1. Registros históricos

El registro histórico de la transacción registra todos los cambios realizados sobre la base de datos y almacena suficiente información para permitir deshacer cualquier cambio (retroceso) o rehacer en el caso de un fallo del sistema o solicitud de retroceso.

El registro histórico es, desde un punto de vista lógico, un flujo potencialmente infinito de registros históricos identificado por números de secuencia del registro histórico (Log Sequence Number, LSN). Desde un punto de vista físico, una porción del flujo se almacena en archivos de registros históricos. Los registros históricos se guardan en los archivos de registros históricos hasta que se realiza una copia de seguridad y no hay necesidad por parte del sistema de retroceso o réplica. Los archivos de registro histórico crecen y disminuyen en tamaño para acomodarse a los registros que se tienen que almacenar. Adicionalmente los archivos de registro histórico se pueden agregar a una base de datos (en nuevos discos, por ejemplo) mientras que el sistema se está ejecutando y sin bloquear ninguna operación y todos los registros históricos se tratan como si fueran un archivo continuo.

27.5.3.2. Recuperación de caídas

El sistema de recuperación de SQL Server tiene muchos aspectos en común con el algoritmo de recuperación ARIES (véase el Apartado 17.9.6), y en este apartado se muestran algunas de las diferencias clave.

SQL Server posee una opción de configuración denominada intervalo de recuperación, que permite a un administrador limitar el tiempo que SQL Server debería tardar en recuperarse después de una caída. El servidor ajusta dinámicamente la frecuencia en los puntos de comprobación para reducir el tiempo de recuperación. Los puntos de comprobación eliminan todas las páginas desfasadas de la memoria intermedia, y se ajustan a las capacidades del sistema E/S y a su carga de trabajo para eliminar de forma efectiva cualquier impacto en las transacciones que se ejecutan.

En el inicio, después de una caída, el sistema inicia varias hebras (dimensionadas automáticamente al número de UCP) para iniciar la recuperación de varias bases de datos en paralelo. La primera fase de la recuperación es un paso de análisis en el registro histórico, que construye una tabla de páginas desfasadas y una lista de transacciones activas. La siguiente fase es un inicio de la fase rehacer desde el último punto de comprobación y realizar todas las operaciones. Durante la fase rehacer se utiliza la tabla de páginas desfasadas para leer anticipadamente las páginas de datos. La fase final es una fase deshacer donde se retroceden todas las transaccio-

nes incompletas. La fase deshacer se divide realmente en dos partes puesto que SQL Server utiliza un esquema de recuperación en dos niveles. Las transacciones en el primer nivel (aquellas que involucran operaciones internas tales como asignación de espacio y divisiones de página) se deshacen primero, seguidas por las transacciones del usuario.

27.5.3.3. *Recuperación de los medios*

Las capacidades de copia de seguridad y restauración de SQL Server permiten la recuperación de muchos fallos, incluyendo la pérdida o corrupción de los medios de disco, errores del usuario y pérdida permanente de un servidor. Además, la copia de seguridad y restauración de las bases de datos es útil para otros propósitos, tales como copiar una base de datos desde un servidor a otro y el mantenimiento de sistemas en espera. Los sistemas en espera se crean con una característica denominada envío del registro histórico que realizan la copia de seguridad continua de los registros históricos de la transacción desde una base de datos origen y entonces copia y restaura dichos registros históricos a una o más bases de datos en espera, manteniendo las bases de datos en espera sincronizadas con la base de datos origen. Los sistemas en espera pueden funcionar en cuestión de minutos. Además, un sistema en espera puede estar disponible para procesamiento de consultas de solo lectura.

SQL Server posee tres modelos de recuperación distintos que los usuarios pueden elegir para cada base de datos. Mediante la especificación de un modelo de recuperación, el administrador declara el tipo de las capacidades de recuperación requeridas (tales como restauración en un punto y envío del registro histórico) y las copias de seguridad requeridas para lograrlos. Se pueden realizar copias de seguridad de bases de datos, archivos, grupos de archivos y del registro histórico de transacciones. Todas las copias de seguridad son difusas y completamente en línea; esto es, no bloquean ninguna operación DML o DDL cuando se ejecutan. Las operaciones de copia de seguridad y restauración están muy optimizadas y limitadas solamente por la velocidad de los medios en los que se realiza la copia de seguridad. SQL Server puede realizar copias de seguridad en disco y en dispositivos de cinta (hasta 64 en paralelo) y tiene interfaces de programación de aplicaciones de copia de seguridad de alto rendimiento para que las usen productos terceros.

27.6. ARQUITECTURA DEL SISTEMA

Un ejemplar de SQL Server es un único proceso del sistema operativo que es también un punto de referencia para las solicitudes de ejecución de SQL. Las aplicaciones interactúan con SQL Server mediante varias bibliotecas en el cliente (como ODBC y OLE-DB) con el fin de ejecutar SQL.

27.6.1. Grupos de hebras en el servidor

Con el fin de minimizar el cambio de contexto en el servidor y para controlar el grado de multiprogramación, el proceso SQL Server mantiene un grupo de hebras que ejecutan solicitudes del cliente. Cuando las solicitudes llegan al cliente se les asigna una hebra sobre la cual se ejecutan. La hebra ejecuta las instrucciones SQL enviadas por el cliente y envía el resultado de vuelta. Una vez que la solicitud del usuario se completa, la hebra se devuelve al grupo de hebras.

Además de las solicitudes del usuario, el grupo de hebras también se utiliza para asignar hebras para tareas internas secundarias.

- Escritor diferido (Lazywriter): esta hebra se dedica a asegurar que una cierta cantidad del grupo de memorias intermedias está libre y disponible siempre para la asignación del sistema. La hebra también interactúa con el sistema operativo para determinar la cantidad óptima de memoria que se debería consumir en el proceso SQL Server.
- Punto de comprobación: esta hebra verifica de forma periódica todas las bases de datos con el fin de mantener un rápido intervalo de recuperación para el inicio de las bases de datos del servidor.
- Monitor de interbloqueo: esta hebra supervisa otras hebras, buscando interbloqueos en el sistema. Es responsable de la detección de interbloqueos y también busca una víctima con el fin de permitir progresar al sistema.

Cuando el procesador de consultas elige un plan paralelo para ejecutar una consulta determinada puede asignar varias hebras que trabajen en nombre de la hebra principal para ejecutar la consulta.

Puesto que la familia de sistemas operativos Windows NT proporciona un soporte nativo para las hebras, SQL Server utiliza hebras NT para su ejecución. Sin embargo, SQL Server se puede configurar para ejecutar hebras en modo usuario además de las hebras del núcleo en sistemas de altas prestaciones para evitar el coste de un cambio de contexto del núcleo en el intercambio de una hebra.

27.6.2. Gestión de la memoria

Hay muchos usos distintos de memoria en el proceso de SQL Server:

- **Grupo de memorias intermedias:** el mayor consumidor de memoria en el sistema es el grupo de

memorias intermedias. El grupo de memorias intermedias mantiene una caché de las páginas de la base de datos más recientemente utilizadas. Emplea un algoritmo de reemplazamiento de reloj con una política de robo sin fuerza, esto es, las páginas de la memoria intermedia con actualizaciones no comprometidas se pueden reemplazar («robar»), y no se fuerza su envío a disco debido a la transacción. Las memorias intermedias también obedecen el protocolo del registro histórico de escritura anticipada para asegurar que la recuperación de las caídas y los medios es correcta.

- **Asignación de memoria dinámica:** es la memoria que se asigna de forma dinámica para ejecutar solicitudes enviadas por el usuario.
- **Caché de planes y ejecución:** esta caché almacena los planes compilados para varias consultas que los usuarios han ejecutado previamente en el sistema. Esto permite que varios usuarios compartan el mismo plan (ahorrando memoria) y también ahorra tiempo de compilación de la consulta para consultas similares.
- **Concesiones de mucha memoria:** para los operadores de consulta que consumen grandes cantidades de memoria tales como reuniones por asociación y ordenaciones.

SQL Server utiliza un elaborado esquema de asignación de memoria para dividir su memoria entre los varios usos descritos arriba. Un único administrador de la memoria gestiona de forma centralizada toda la memoria utilizada por SQL Server. El administrador de memoria es responsable de la división y distribución de la memoria de forma dinámica entre los diversos consumidores de memoria del sistema. Distribuye esta memoria de acuerdo con un análisis del coste relativo de memoria para cualquier uso particular.

El administrador de memoria interactúa con el sistema operativo para decidir de forma dinámica cuánta memoria se debería consumir de la cantidad total de memoria en el sistema. Esto permite que SQL Server sea bastante agresivo en el uso de la memoria en el sistema pero también devuelve la memoria al sistema cuando otros programas la necesita sin causar excesivos fallos de página.

27.7. ACCESO A DATOS

Este apartado describe las Interfaz de programación de aplicaciones (Application Programming Interface, API) de acceso a datos soportadas por SQL Server y cómo se utilizan estas API para las comunicaciones entre los componentes internos del servidor.

27.7.1. API de acceso a datos

SQL Server soporta una serie de interfaces de programación de aplicaciones (API) de acceso a datos, incluyendo:

- Objetos de datos ActiveX (ActiveX Data Objects, ADO)
- OLE-DB
- Conectividad abierta de bases de datos (ODBC, Open Database Connectivity) y las API construidas sobre ODBC: objetos de datos remotos (Remote Data Objects, RDO) y objetos de acceso a datos (Data Access Objects, DAO)
- SQL incorporado para C (Embedded SQL, ESQL)
- La biblioteca de bases de datos heredadas para C, que fue desarrollada específicamente para ser usada con versiones anteriores de SQL Server que preceden a la norma SQL-92
- HTTP y URLs

Las aplicaciones Internet pueden utilizar URL que especifican las rutas virtuales del servidor de información de Internet (IIS, Internet Information Server) que hacen referencia a una ejemplar de SQL Server. Un URL puede contener una consulta XPath, una instrucción Transact-SQL o una plantilla XML. Además de utilizar URL, las aplicaciones Internet también pueden utilizar ADO o OLE-DB para trabajar con datos en la forma de documentos XML.

Los desarrolladores de aplicaciones utilizan API tales como ADO y ODBC para acceder a las capacidades de SQL Server desde la capa intermedia. Algunas API tales como OLE-DB se utilizan internamente para integrar componentes del núcleo del servidor y permitir la réplica y el acceso distribuido para SQL y otros orígenes externos.

27.7.2. Comunicación dentro de SQL Server

El servidor de la base de datos de SQL Server tiene dos partes principales: el motor relacional (MR) y el motor de almacenamiento (MA), como se muestra en la Figura 27.7. La arquitectura de SQL Server claramente separa los componentes del motor relacional y de almacenamiento con el servidor y estos componentes utilizan interfaces OLE-DB para comunicarse entre sí.

El procesamiento de una instrucción **select** que hace referencia solamente a tablas en bases de datos locales se puede describir como sigue. El motor relacional compila la instrucción **select** en un plan de ejecución optimizado. El plan de ejecución define una serie de operaciones sobre conjuntos de fila sencillos desde las tablas individuales o índices referenciados en la instrucción

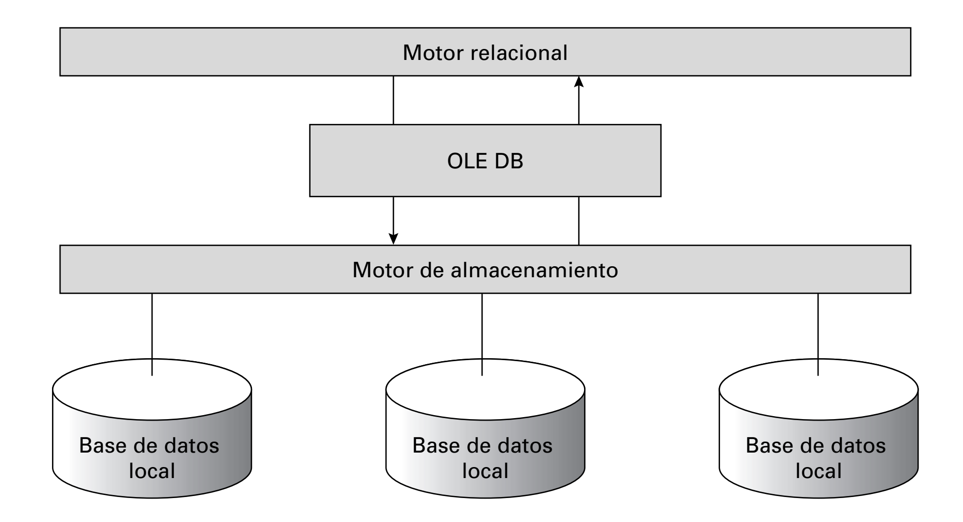


FIGURA 27.7. OLE-DB como una interfaz del sistema interno de gestión de bases de datos.

select. El conjunto de filas solicitado por el motor relacional devuelve la cantidad de datos que una tabla o índice necesita para ejecutar las operaciones necesarias para construir el conjunto de resultados **select**. Por ejemplo, la siguiente consulta calcula el resultado de dos índices:

select NombreEmpresa, IDPedido, FechaEnvío
from Clientes **as** Cli **join** Pedidos **as** Ped
on (Cli.IDCliente = Ped.IDCliente)

El motor relacional solicita dos conjuntos de filas, uno para el índice agrupado en clientes y el otro en un índice no agrupado en pedidos. El motor relacional entonces utiliza las interfaces OLE-DB para solicitar que el motor de almacenamiento abra los conjuntos de filas con estos índices. Puesto que el motor relacional funciona según los pasos del plan de ejecución y necesita datos, utiliza OLE-DB para extraer las filas individuales de los conjuntos de filas. El motor de almacenamiento transfiere los datos desde las memorias intermedias de los datos hasta los operadores de ejecución de la consulta, la cual se ejecuta en el motor relacional. El motor relacional combina los datos de los conjuntos de filas del motor de almacenamiento en el conjunto de resultados final transmitido al usuario. Esta comunicación entre el motor relacional y el motor de almacenamiento mediante la interfaz OLE-DB permite al motor relacional procesar las consultas con cualquier origen de datos que exponga tales interfaces. Dichos orígenes de datos pueden ser otros sistemas SQL Server (consultas distribuidas) y otros proveedores de datos OLE-DB relacionales y no relacionales (consultas heterogéneas).

27.8. DISTRIBUCIÓN Y RÉPLICAS

Este apartado describe las capacidades de SQL Server para procesamiento de consultas distribuido y heterogéneo, así como las réplicas.

27.8.1. Procesamiento de consultas distribuidas heterogéneas

La capacidad de consultas distribuidas heterogéneas de SQL Server permite las consultas transaccionales y actualizaciones en una serie de orígenes relacionales y no relacionales mediante proveedores de datos OLE-DB que se ejecutan en una o más computadoras. SQL Server soporta dos métodos para hacer referencia a orígenes de datos OLE-DB heterogéneos en instrucciones Transact-SQL.

El método nombres del servidor vinculados utiliza procedimientos almacenados del sistema para asociar el nombre de un servidor con un origen de datos OLE-DB. Los objetos en estos servidores vinculados se pueden referenciar en instrucciones Transact-SQL utilizando el convenio de nombres de cuatro parte descrito más adelante. Por ejemplo, si el nombre de un servidor vinculado de *ServSQLDept* se define en otra copia de SQL Server, la siguiente instrucción referencia una tabla en ese servidor:

select *
from ServSQLDept.Northwind.dbo.Employees

En SQL Server se registra un origen de datos OLE-DB como un servidor vinculado. Una vez que se defi-

ne un servidor vinculado se puede acceder a sus datos utilizando el nombre de cuatro partes <servidor vinculado>.<catálogo>.<esquema>.<objeto>. El siguiente ejemplo establece un servidor vinculado a un servidor Oracle mediante un proveedor OLE-DB para Oracle:

```
exec sp_addlinkedserver ServOra, 'Oracle 7.3',
        'MSDAORA', 'OracleServer'
```

Una consulta en este servidor vinculado se expresa como:

```
select *
from ServOra.CORP.ADMIN.VENTAS
```

Además, SQL Server soporta funciones incorporadas parametrizadas de tipo tabla denominadas **openrowset** y **openquery**, que permiten enviar consultas no interpretadas a un proveedor o servidor vinculado, respectivamente, en el dialecto soportado por el proveedor. La siguiente consulta combina la información almacenada en un servidor Oracle y un Microsoft Index Server. Lista todos los documentos y sus autores que contiene las palabras Data y Access ordenadas por el departamento y nombre del autor.

```
select e.dept, f.AutorDoc, f.NombreArchivo
from ServOra.Corp.Admin.Empleados e,
openquery(ArchivosEmp,
           'select AutorDoc, NombreArchivo
           from scope(«c: \ EmpDocs»)
           where contains(' «Datos» near()
           «Access»')>0') as f
where e.nombre = f. AutorDoc
order by e.dept, f.AutorDoc
```

También se puede especificar el nombre del servidor vinculado en una instrucción **openquery** para abrir un conjunto de filas desde el origen de datos OLE-DB. El conjunto de filas se puede entonces referenciar como una tabla en instrucciones Transact-SQL. El método ad hoc de nombres de conectores se utiliza para referencias infrecuentes a un origen de datos. Este método utiliza una función de tipo tabla denominada **openrowset**, donde la información necesaria para conectarse a los orígenes de datos se proporciona como argumentos a la función. El conjunto de filas se puede entonces referenciar como se referencia a una tabla en instrucciones SQL. Por ejemplo, la siguiente consulta accede a las exploraciones *employees* almacenadas en tablas en la base de datos *Northwind* en un proveedor de datos de Microsoft Access. Nótese que aunque el enfoque de nombre de servidor vinculado descrito anteriormente encapsula toda la información necesaria para conectarse a un origen de datos, el enfoque ad-hoc de conectores requiere que el usuario especifique el nombre del proveedor de datos (Microsoft.Jet.OLE-DB.4.0), el nombre completo de la ruta de acceso del archivo de datos, el identificador del usuario, la contraseña (vacío en el ejemplo de abajo) y el nombre de la tabla a la que se accede.

```
select *
from openrowset('Microsoft.Jet.OLE DB.4.0',
           'c:\Ejemplos\
           Northwind.mdb';'Admin';' ';
           Employees)
```

El motor relacional utiliza las interfaces OLE-DB para abrir los conjuntos de filas sobre los servidores vinculados, para extraer las filas y para gestionar las transacciones. Para cada origen de datos OLE-DB al que se accede como un servidor vinculado, debe estar presente un proveedor OLE-DB sobre el servidor en el que se ejecuta SQL Server. El conjunto de operaciones Transact-SQL que se pueden utilizar en un origen de datos OLE-DB específico depende de las capacidades del proveedor OLE-DB. Siempre que es efectivo en el coste, SQL Server envía las operaciones relacionales tales como las reuniones, restricciones, proyecciones, ordenaciones y agrupaciones mediante operaciones al origen de datos OLE-DB.

SQL Server utiliza el coordinador de transacciones distribuidas de Microsoft (Microsoft Distributed Transaction Coordinator) y las interfaces de transacción de OLE-DB del proveedor para asegurar la atomicidad de las transacciones sobre varios orígenes de datos.

Veamos un escenario típico para el uso de consultas distribuidas. Consideremos una gran compañía de seguros que tiene empresas subsidiarias en varios países. Cada oficina regional selecciona el producto que almacena sus datos de ventas. La subsidiaria del Reino Unido almacena sus datos en Oracle, la subsidiaria en Australia almacena sus datos en Microsoft Access y la subsidiaria de España almacena sus datos en Microsoft Excel y la subsidiaria de Estados Unidos almacena sus datos en SQL Server. Un ejecutivo de ventas internacional desea un informe que liste, de forma trimestral por los últimos tres años, las directivas de seguros, las subsidiarias y los representantes de ventas con las ventas más altas en cada cuatrimestre. Cada una de estas tres consultas se puede realizar mediante una única consulta distribuida, que se ejecuta en SQL Server.

27.8.2. Réplica

La réplica de SQL Server proporciona un conjunto de tecnologías para copiar y distribuir los datos y objetos de la base de datos de una base de datos a otra y también mantener la sincronización entre las bases de datos.

La réplica puede recoger datos corporativos desde sitios geográficamente dispersos para propósitos de informes y diseminar los datos a usuarios remotos en una red de área local o usuarios móviles sobre conexiones telefónicas o Internet. La réplica de Microsoft SQL Server también mejora el rendimiento de las aplicaciones dimensionando para mejorar el rendimiento

total de lectura entre réplicas, como es común al proporcionar servicios de caché de datos de la capa intermedia para sitios Web al tiempo que se mantiene la consistencia transaccional en el conjunto de datos duplicado.

27.8.2.1. Modelo de réplicas

SQL Server introdujo la metáfora *Publicar-Suscribir* para la réplica de la base de datos y extiende esta metáfora de la industria editorial a todas sus herramientas de administración de réplicas y supervisión.

- El publicador es un servidor que hace los datos disponibles para la réplica a otros servidores. El publicador puede tener una o más publicaciones, cada una representando un conjunto de datos y objetos de la base de datos relacionados lógicamente. Los objetos discretos en una publicación, incluyendo tablas, procedimientos almacenados, funciones definidas por el usuario, vistas, vistas materializadas y más, todos llamados artículos. La agregación de un artículo a una publicación permite la personalización extensiva de la forma en la que se duplica el objeto, incluyendo el nombre y propietario de los objetos destino, restricciones sobre las cuales los usuarios se pueden suscribir para recibir sus datos y cómo se debería filtrar el conjunto de resultados. Por ejemplo, una tabla puede tener su conjunto de datos completo o un subconjunto de sus datos dividido horizontalmente haciéndolos disponibles para la distribución mediante su definición como un artículo. Todos los formularios de la réplica de SQL Server soportan la división horizontal y vertical desde una tabla publicada.
- Los suscriptores son servidores que reciben los datos replicados de un publicador. Los suscriptores pueden suscribir convenientemente a solamente las publicaciones que requieren de uno o más publicadores sin considerar el número o tipo de opciones de réplica que implemente cada uno. Dependiendo del tipo de opciones de réplica seleccionado, el suscriptor se puede utilizar como una réplica de sólo lectura o se pueden realizar cambios en los datos que se propagan automáticamente al publicador y, por consiguiente, al resto de réplicas. Las réplicas de SQL Server soportan suscripciones de inserción y extracción verdaderas; esto es, la programación o iniciación de acciones de sincronización se pueden controlar por el publicador o mediante sus suscriptores según requieran las necesidades del negocio. Los suscriptores también pueden volver a publicar los datos a los que se suscriben, dando soporte a una topología de réplica tan flexible como requiera la empresa.
- El distribuidor es un servidor que alberga la base de datos de distribución y almacena algunos metadatos de réplica. La función del distribuidor varía, dependiendo de las opciones de réplica seleccionadas.

27.8.2.2. Opciones de réplica

La réplica de Microsoft SQL Server ofrece un amplio espectro de elecciones de tecnología. Para decidir sobre las opciones de réplica apropiadas a utilizar, un diseñador de bases de datos debe determinar las necesidades de la aplicación con respecto a la operación autónoma del sitio involucrado y el grado de consistencia transaccional requerido.

- La réplica instantánea copia y distribuye los datos y objetos de la base de datos exactamente como aparecen en un instante del tiempo. La réplica instantánea no requiere un seguimiento continuo de los cambios, puesto que los cambios no se propagan de forma incremental a los suscriptores. Los suscriptores se actualizan con una actualización completa del conjunto de datos definido por la publicación de una forma periódica. Las opciones disponibles con la réplica instantánea pueden filtrar los datos publicados y pueden permitir que los suscriptores modifiquen los datos replicados y propaguen dichos cambios al publicador.
- Con la publicación transaccional el publicador propaga una instantánea de datos a los suscriptores, entonces envía las modificaciones de datos incrementales a los suscriptores como transacciones e instrucciones discretas. El seguimiento del cambio incremental ocurre dentro del motor del núcleo de SQL Server, que marca las transacciones que afectan a los objetos replicados en el registro histórico de la base de datos publicadora. Un agente de réplica lee estas transacciones del registro histórico de la base de datos, aplica cualquier lógica de división y las almacena en la base de datos de la distribución, que actúa como una cola fiable que soporta el mecanismo de almacenamiento y envío de la réplica transaccional. (Las colas fiables son lo mismo que las colas duraderas descritas en el Apartado 24.1.1.) Otro proceso de réplica, el agente de distribución que se ejecuta o en el distribuidor (inserción) o el suscriptor (extracción), entonces envía los cambios a cada suscriptor. Al igual que la réplica instantánea, la réplica transaccional ofrece a los suscriptores la opción de realizar actualizaciones inmediatas que utilicen un compromiso de dos fases que reflejen aquellos cambios de forma consistente en el publicador y suscriptor.
- La réplica por mezcla permite que cada réplica en la empresa funcione con total autonomía en conexión o sin conexión. El sistema supervisa los metadatos según los cambios sobre objetos publicados en los publicadores y suscriptores de todas las bases de datos duplicadas y el agente de réplica mezcla estas modificaciones de los datos durante la sincronización entre los pares replicados y asegura la convergencia de los datos mediante una detección

y resolución automática del conflicto. El agente de réplica utilizado en el proceso de sincronización incorpora numerosas opciones de políticas de resolución de conflictos, y la resolución de conflictos personalizada se puede escribir mediante el uso de procedimientos almacenados o mediante el uso de una interfaz del **modelo de objetos componente (Component Object Model, COM)** extensible.

27.9. CONSULTAS DE TEXTO COMPLETO SOBRE DATOS RELACIONALES

La capacidad de texto completo en SQL Server de Microsoft soporta la creación y mantenimiento de índices de texto completo sobre cadenas de caracteres y columnas de imágenes almacenadas dentro de las tablas SQL Server, así como búsquedas de texto completo basadas en estos índices. La capacidad de texto completo se implementa mediante el servicio Microsoft Search, desarrollado independientemente de SQL Server, para permitir las búsquedas de texto completo en los datos del sistema de archivos. El primer paso hacia la integración del servicio de búsqueda con SQL Server fue transformar el servicio de búsqueda en un proveedor OLE-DB. Este paso permitía escribir las aplicaciones en lenguajes tales como Visual Basic y C++ con acceso a los datos almacenados en el sistema de archivos mediante el uso de ADO, y también proporcionaba la capacidad de conectar el proveedor de texto completo en SQL Server como un origen de datos heterogéneo. El segundo paso involucraba una integración débilmente acoplada entre SQL Server y el servicio de búsqueda para permitir el indexado del texto completo del contenido de la tabla. Esta integración está débilmente acoplada en el sentido en que los índices de texto completo se almacenan en el sistema de archivos fuera de la base de datos. La Figura 27.8 ilustra la arquitectura general de esta integración.

Hay dos aspectos para el soporte de texto completo: (1) creación de índices y mantenimiento y (2) soporte de la consulta. El soporte de indexado involucra la creación, actualización y administración de catálogos de texto completo e índices definidos para una tabla o tablas en la base de datos. El soporte de la consulta involucra el procesamiento de las consultas de búsqueda de texto completo. Dado un predicado de texto completo, el servicio de búsqueda determina las entradas en el índice que cumplen el criterio de selección de texto completo. Para cada entrada que cumple el criterio de selección, el componente de consulta del servicio de búsqueda devuelve una fila, en un conjunto de filas OLE-DB, conteniendo la identidad de la fila cuyas columnas coinciden con el criterio de búsqueda y un valor de clasificación. Este conjunto de filas se utiliza como entrada a la consulta que está siendo procesada por el motor relacional SQL, al igual que cualquier otro conjunto de filas originado

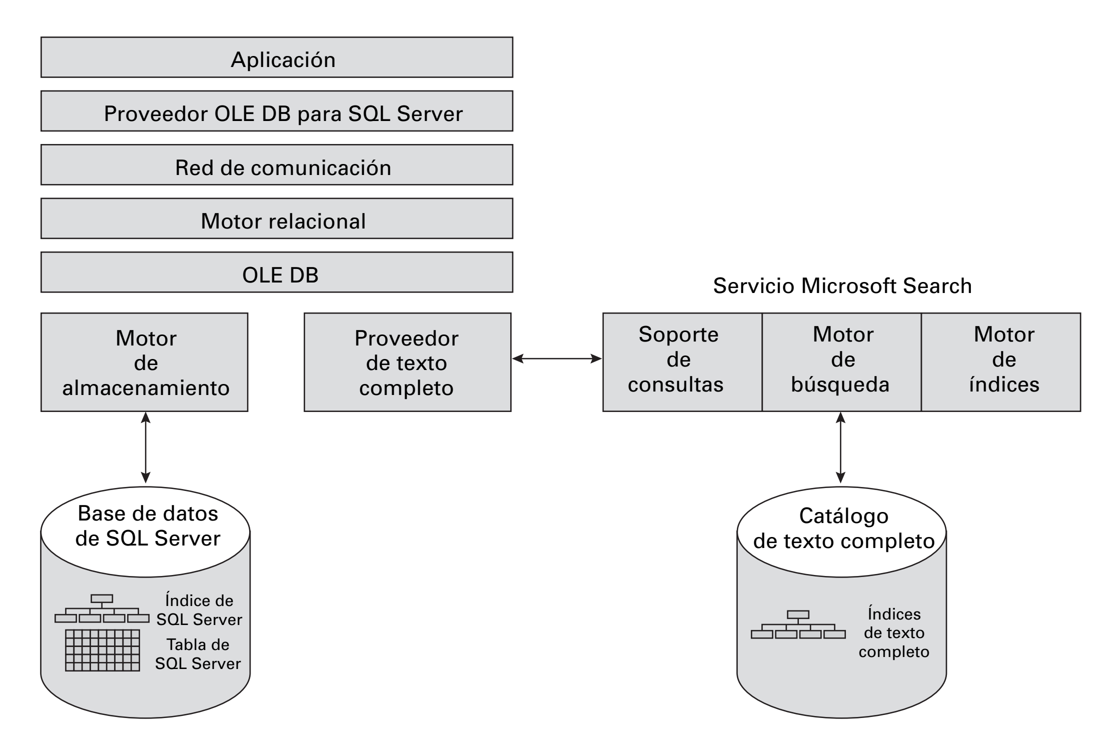


FIGURA 27.8. Integración de un componente de texto completo con un SGBD relacional.

de tablas o índices dentro del servidor. El motor relacional combina este conjunto de filas con la tabla base sobre la identidad de la fila y evalúa el plan de ejecución que genera el conjunto de resultados final. Los tipos de consultas de texto completo que soporta este esquema incluye la búsqueda de palabras o frases, palabras parecidas entre sí y formas derivadas de verbos y nombres.

Los catálogos de texto completo e índices no se almacenan en una base de datos SQL Server. Se almacenan en archivos separados gestionados por el servicio Microsoft Search. Los archivos del catálogo de texto completo no se recuperan durante la actividad de recuperación de SQL Server, y no se puede realizar una copia de seguridad y restauración mediante el uso de las instrucciones **backup** y **restore**. Se deben volver a sincronizar los catálogos de texto completo separadamente después de una operación de recuperación o restauración.

27.10. ALMACENES DE DATOS Y SERVICIOS DE ANÁLISIS

Las aplicaciones de las bases de datos requieren la transformación de datos de muchos orígenes en un conjunto cohesivo y consistente de datos configurados apropiadamente para su uso en las operaciones de los almacenes de datos. SQL Server 2000 proporciona una herramienta para tales tareas, los servicios de transformación de datos (Data Transformation Services, DTS). DTS puede acceder a los datos desde una amplia variedad de orígenes y transformarlos mediante el uso de especificaciones de transformación incorporadas y personalizadas.

Los Servicios de análisis de SQL Server proporcionan un rápido acceso al almacén de datos. Los datos del almacén de datos se extraen, resumen, organizan y almacenan en estructuras multidimensionales para una rápida respuesta a consultas de usuarios finales. Los servicios de análisis también proporcionan una arquitectura para el acceso a la recopilación de datos. Estos datos también se pueden enviar al cliente en una forma multidimensional o relacional.

Este apartado describe brevemente DTS y el procesamiento en conexión analítico (Online Analytical Processing, OLAP) de los servicios de análisis.

27.10.1. Servicios de transformación de datos

El almacén de datos es un enfoque para gestionar los datos en los cuales los orígenes de datos heterogéneos (normalmente varias bases de datos OLTP) migran a una base de datos homogénea separada. Los almacenes de datos proporcionan muchos beneficios a los usuarios logísticos. Los datos se organizan para facilitar consultas analíticas en lugar de procesamiento de transacciones. Se pueden resolver las diferencias entre las estructuras de datos en varias bases de datos heterogéneas. Las reglas de transformación de datos se pueden aplicar para validar y consolidar los datos cuando éstos se mueven desde la base de datos OLTP operacional al almacén de datos. Los aspectos de seguridad y rendimiento se pueden resolver sin requerir cambios en los sistemas de producción.

Los servicios de transformación de datos (Data Transformation Services, DTS) de SQL Server proporcionan la funcionalidad para importar, exportar y transformar los datos entre varios orígenes de datos heterogéneos de forma interactiva o automáticamente según una planificación regular. Se accede de forma uniforme a todos los orígenes de datos mediante los proveedores OLE-DB. Las secuencias de órdenes (guiones) ejecutan tareas de transformación entre orígenes de datos fuente y destino. La extracción, transformación y proceso de carga involucra en los sistemas operacionales la validación de los datos, migración de los datos, normalización de los datos a un dominio común y transformaciones de datos para asignar o resumir los valores. Una actividad DTS se organiza en un paquete que incluye tres componentes: (1) los objetos de conexión definen cada origen de datos OLE-DB fuente o destino, (2) los objetos tarea definen las acciones específicas a ejecutar y (3) los objetos paso definen la secuencia en la cual se ejecutan las tareas. Los pasos también definen si la ejecución de una tarea es dependiente de los resultados de una tarea anterior. Data Pump de DTS es un componente de servicio OLE-DB multienhebrado que proporciona la infraestructura para importar, exportar y transformar los datos entre orígenes de datos OLE-DB heterogéneos. Las tareas Data Pump de DTS permiten la invocación de programas del usuario que resuelven correspondencias complejas entre las columnas de origen y destino mientras se transfieren los datos.

El procesamiento mediante una tarea Data Pump de DTS (Figura 27.9) incluye la conexión a los objetos de conexión origen y destino, determinando las propiedades del conjunto de filas origen (que se construye mediante los formatos de columna de la tabla origen o el resultado de ejecutar una consulta) y pasando esta información y una definición de todas las transformaciones especificadas a Data Pump de DTS. Durante la ejecución, Data Pump de DTS extrae las filas del origen de datos y copia las columnas origen a la columna destino según se define en las correspondencias de transformación encapsuladas en las secuencias de órdenes del modelo de objetos componentes (COM). Cada fila origen transformada se inserta en el origen de datos destino OLE-DB.

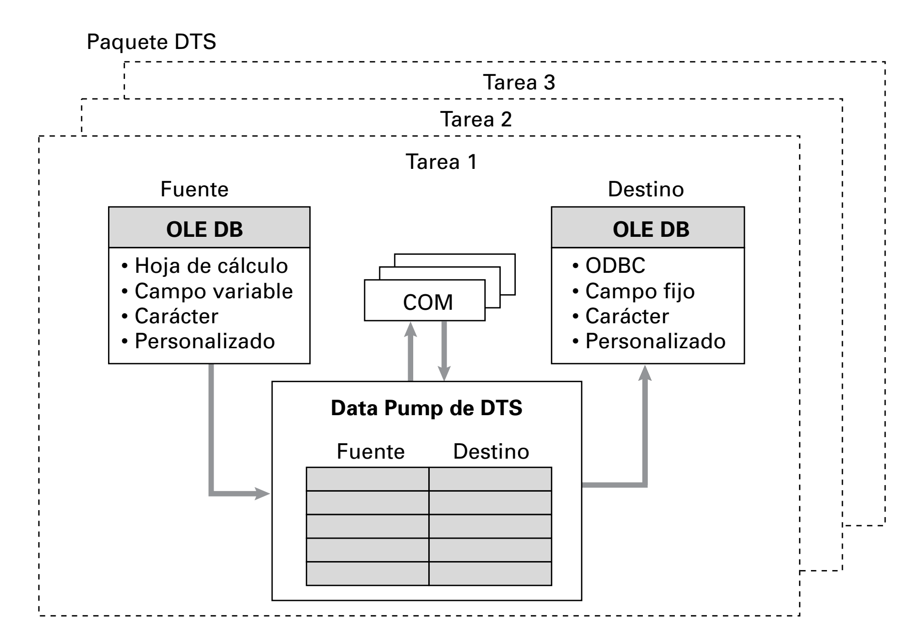


FIGURA 27.9. DTS accede a varios orígenes de datos mediante OLE-DB.

27.10.2. Servicios de procesamiento analítico en línea

Los servicios OLAP de SQL Server organizan los datos de un almacén de datos en cubos multidimensionales con información resumida precalculada para proporcionar respuestas eficientes a consultas analíticas complejas. El objeto principal de OLAP es el cubo, una representación multidimensional de los datos detallados y resumidos. Un cubo consiste en un origen de datos, dimensiones, medidas y divisiones. Un almacén de datos puede soportar muchos cubos distintos. Las consultas multidimensionales en los cubos devuelven objetos de tipo conjunto de datos.

Los servicios OLAP proporcionan capacidades cliente y servidor para crear y gestionar datos OLAP multidimensionales (Figura 27.10). Las operaciones del servidor incluyen la creación de cubos de datos multidimensionales a partir de bases de datos de almacén de datos relacionales y cubos almacenados en estructuras de cubo multidimensionales, en bases de datos relacionales, y en combinaciones de ambos. Los metadatos de estructuras cubo multidimensionales se almacenan en un depósito en una base de datos relacional. Las operaciones del cliente son proporcionadas por el servicio de tablas dinámicas, que es un proveedor OLE-DB que soporta OLE-DB para extensiones de la interfaz OLAP. El servicio de tablas dinámicas es un servidor OLAP en el proceso diseñado para proporcionar análisis de datos en conexión y sin conexión y acceso en conexión a datos OLAP. El servicio de tablas dinámicas funciona como un cliente de los servicios OLAP.

27.11. XML Y SOPORTE WEB

Muchas aplicaciones se construyen actualmente como sistemas distribuidos débilmente acoplados donde los componentes individuales (frecuentemente llamados servicios) se combinan entre sí. Puesto que muchos de estos componentes se reutilizarán para otras aplicaciones, la arquitectura necesita ser lo suficientemente flexible como para permitir que los componentes individuales se unan o abandonen el conglomerado heterogéneo de servicios y componentes y que cambien su diseño interno y modelos de datos sin arriesgar toda la arquitectura.

XML y el soporte Web de SQL Server simplifica la construcción de componentes basados en la base de datos y los servicios que utilizan XML para la capa de integración. Esta capa oculta la heterogeneidad entre los componentes y proporciona el pegamento que permite a los componentes individuales tomar parte en el sistema débilmente integrado.

SQL Server proporciona mecanismos para producir XML a partir de datos relacionales y para consumir XML y hacerlo corresponder con datos relacionales. Junto con el componente de acceso HTTP en las herramientas de acceso al cliente, esta posibilidad permite que SQL Server se utilice como el proveedor de datos y los componentes consumidores de sitios y servicios Web.

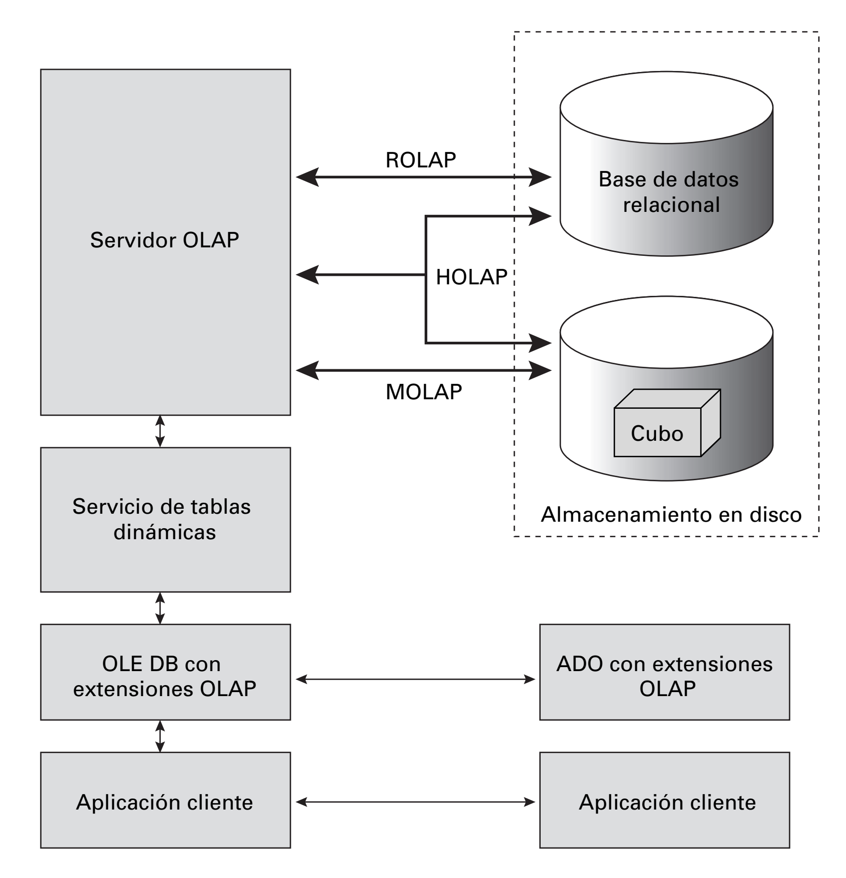


FIGURA 27.10. Integración de un servidor OLAP y un SGBD relacional.

27.11.1. Arquitectura del acceso XML a SQL Server

La Figura 27.11 muestra un diagrama arquitectónico de bloques de alto nivel del soporte XML de SQL Server. Puesto que las distintas aplicaciones aplican su lógica de negocios en posiblemente distintas ubicaciones, la arquitectura proporciona acceso HTTP directo cuando solamente se necesita ejecutar la visualización usando XSLT en la capa intermedia y el resto del procesamiento de la lógica del negocio se puede insertar completamente en el cliente o en el servidor de la base de datos. Para arquitecturas de dos capas o donde la lógica del negocio se tiene que ejecutar en la capa intermedia se utiliza frecuentemente un acoplamiento más estrecho de la lógica del negocio al acceso de la base de datos por razones de rendimiento y programabilidad. Por ello, todos los accesos a las características XML es a través del proveedor SQL OLE-DB; esto se aplica al acceso ADO y también al acceso HTTP mediante la extensión ISAPI al Internet Information Server (IIS).

Existen varias formas de acceder a SQL Server mediante HTTP. ISAPI de SQL Server está registrado con IIS para gestionar los mensajes a una raíz virtual determinada (vroot). ISAPI recibe las solicitudes para esa vroot particular y después de ejecutar la autorización pasa las órdenes apropiadas a través del proveedor SQL OLE-DB a la base de datos. La raíz virtual, como parte del URL, proporciona un mecanismo de abstracción que encapsula el servidor de bases de datos al que se accede y a los ejemplares de la base de datos, los derechos de acceso y los métodos de acceso habilitados. Los métodos de acceso principales proporcionados por SQL Server son el acceso con plantillas y el acceso a vistas XML.

Las plantillas son documentos XML que proporcionan una consulta parametrizada y un mecanismo de actualización a la base de datos. Puesto que oculta la consulta real (o actualización) del usuario proporciona el nivel de desacoplamiento que hace posible la construcción de sistemas débilmente acoplados. Los elementos que contienen consultas son procesados por el procesador de plantillas y utilizados para devolver datos de la base de datos como parte del documento XML resultante. Los elementos no reconocidos por el procesador de la plantilla se devuelven sin modificar. Las plantillas pueden contener instrucciones Transact-SQL, updategrams (véase el Apartado 27.11.3), consultas XPath o una combinación de éstas.

Las vistas XML se definen anotando un documento esquema SML con la correspondencia con las tablas y columnas relacionales. Las jerarquías se corresponden desde y a la base de datos utilizando una anotación de relación que expresa la reunión externa entre el padre y los hijos. Esta vista se puede entonces utilizar para consultar en el lenguaje de navegación de la base de

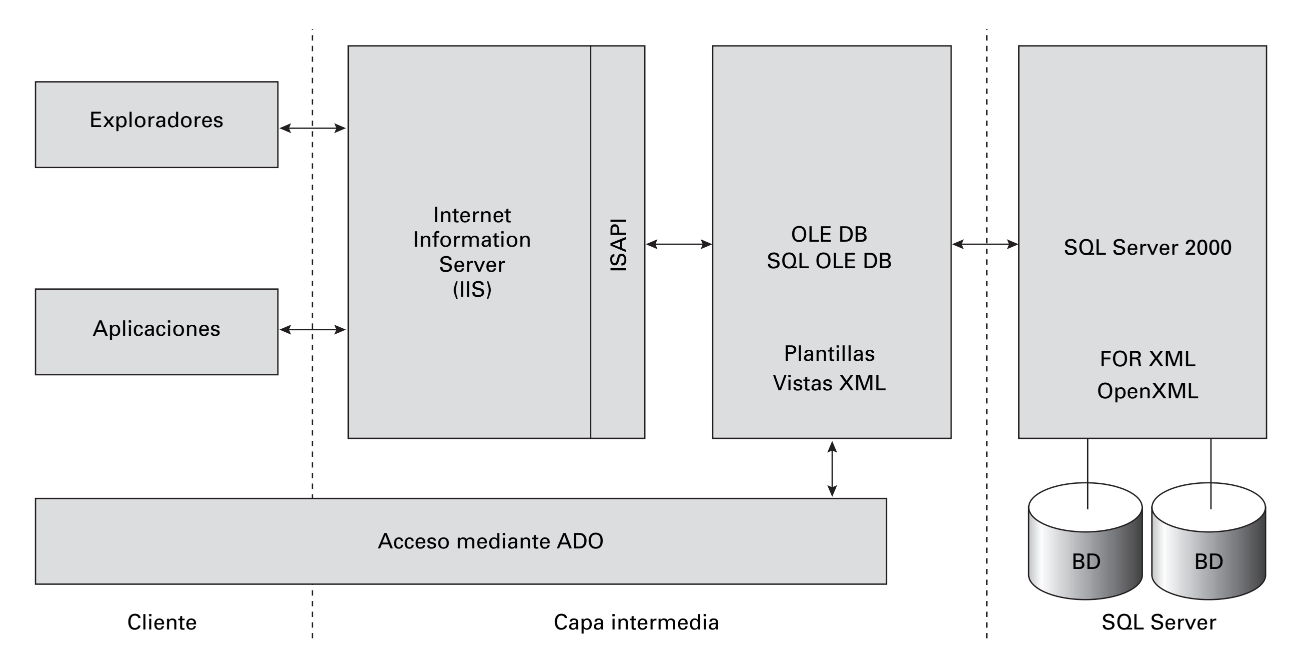


FIGURA 27.11. Visión general de la arquitectura del acceso XML a SQL Server.

datos XPath 1.0 y para actualizarlo mediante el uso de updategrams. Estas características XML (así como las consultas XML del servidor) también son accesibles mediante SQL OLE-DB. Puesto que los datos XML, al contrario que los datos relacionales, no se representan como un conjunto de fijas sino como un flujo marcado de datos, el proveedor OLE-DB de SQL Server se ha extendido con una interfaz de flujo para exponer los resultados XML.

En el lado del servidor, SQL Server proporciona extensiones a la instrucción **select** que simplifica la transformación de datos relacionales en XML y un mecanismo para cortar conjuntos de filas de documentos XML y por ello proporciona una vista relacional sobre los datos XML.

27.11.2. Secuencias de resultados SQL en XML

Las personas familiarizadas con la escritura de consultas de selección SQL pueden necesitar poder generar XML fácilmente a partir de sus resultados de la consulta. Por desgracia, hay muchas formas diferentes en las cuales se puede realizar dicha generación de XML. SQL Server por consiguiente proporciona tres modos distintos para esto con niveles distintos de complejidad y capacidad de autoría XML. Los tres modos se proporcionan mediante una nueva cláusula **select** denominada FOR XML.

Los tres modos son: *raw* (sin formato), *auto* (automático) y *explicit* (explícito). La siguiente instrucción muestra un ejemplo de una consulta en modo *auto*:

```
select Clientes.IDCliente, IDCliente
from Clientes left outer join Pedidos
on Clientes.IDCliente = Pedidos.IDCliente
ordered by Clientes.IDCliente
for XML auto
```

Los tres modos asignan filas a elementos y valores de columna a atributos. La directiva opcional *elements* en el modo *auto* cambia la correspondencia de todos los valores de columna a subelementos. Los modos *raw* y *auto* permiten la generación sencilla de XML a partir de consultas relacionales existentes. El modo *explicit* proporciona control completo de columnas sobre el XML generado tomando un formulario especial de un conjunto de resultados relacional (denominado formato de tabla universal) y transformándolo a XML. Los tres modos canalizan los datos y así se evitan construcciones costosas de documentos en el servidor.

27.11.3. Vistas XML de datos relacionales

El apartado anterior presentaba el enfoque centrado en SQL para generar XML. SQL Server también proporciona un mecanismo que permite la definición de vistas XML virtuales de la base de datos relacional, la cual se puede consultar y actualizar con herramientas basadas en XML. El mecanismo central para proporcionar vistas XML sobre datos relacionales es el concepto de esquema anotado. Los esquemas anotados consisten en una descripción de esquema basada en XML de la vista XML expuesta (en el esquema del lenguaje XML-Data Reduced o W3C XML) y en anotaciones que describen las correspondencias de las construcciones del esquema XML en las construcciones del esquema relacional. Para simplificar la definición de las anotaciones cada esquema proporciona la correspondencia predeterminada si

no hay presentes anotaciones. Las correspondencias predeterminadas asignan un atributo o un subelemento no complejo (uno cuyo tipo de contenido es solamente texto) a una columna relacional con el mismo nombre. El resto de elementos se corresponde con filas de una tabla o vista con el mismo nombre. Las jerarquías se expresan con anotaciones. Una herramienta visual descargable denominada SQL XML View Mapper proporciona una forma para especificar gráficamente las correspondencias y, por tanto, las anotaciones. El esquema anotado no recupera en sí mismo ningún dato sino que solamente define una vista virtual proyectando una vista XML en tablas relacionales. Se puede requerir ahora la vista en un lenguaje de consulta XML y actualizado en un lenguaje de actualización basado en XML.

Las actualizaciones son soportadas por los denominados *updategrams* de XML. Los *updategrams* proporcionan una forma intuitiva de ejecutar una transformación basada en el ejemplar desde un estado anterior a un estado posterior mediante el uso de un control de la concurrencia optimista.

27.11.4. Vistas relacionales de XML

En muchos casos los datos se enviarán al servidor de la base de datos en la forma de un mensaje XML que se tiene que integrar con los datos relacionales después de que se ejecute algo de lógica del negocio opcional sobre los datos dentro de un procedimiento almacenado en el servidor. Esto requiere el acceso mediante programación a los datos XML desde un procedimiento almacenado. Por desgracia, ni DOM ni SAX (véase el Capítulo 10) proporcionan una API adecuada para tratar una vista relacional con los datos XML; esto es, necesita permitir al programador SQL descomponer un mensaje XML en distintas vistas relacionales.

SQL Server proporciona mecanismo de conjuntos de filas en XML mediante el proveedor de conjuntos de filas OpenXML. La vista del conjunto de filas utiliza una expresión XPath (el patrón de filas) para identificar los nodos en el árbol del documento XML que se corresponderá con las filas y utiliza una expresión XPath relativa (el patrón de columnas) para identificar los nodos que proporcionan los valores para cada columna. El proveedor de conjuntos de filas OpenXML puede aparecer en cualquier lugar en una expresión SQL donde un conjunto de filas puede aparecer como un origen de datos. En particular, puede aparecer en la cláusula **from** de cualquier selección.

Una de las ventajas de esta API orientada a conjuntos de filas para los datos XML es que incluye sobre el modelo relacional existente para su uso con XML y proporciona un mecanismo para actualizar la base de datos con datos en formato XML. El uso de XML en conjunción con OpenXML permite actualizaciones multifila con una única llamada a procedimiento almacenado y actualizaciones sobre varias tablas mediante la explotación de la jerarquía XML. Además permite la formulación de consultas que combinan las tablas existentes con los datos XML proporcionados.

27.12. RESUMEN

SQL Server de Microsoft es un paquete completo de gestión de datos que incluye servidor de base de datos relacional, búsqueda e indexación de texto completo, importación y exportación de datos XML, integración de datos distribuidos y heterogéneos, servidor de análisis y cliente para OLAP y recopilación de datos, réplicas entre almacenes de datos heterogéneos, un motor de transformación de datos programable y más. Por ello, SQL Server sirve como fundamento de la familia de Microsoft de productos servidores empresariales.

Durante el tiempo en el que Microsoft ha tenido un control total sobre el código base (después de adquirirlo a Sybase) ha actualizado el código base e integrado las últimas investigaciones prácticas en el producto. SQL Server 2000 (lanzado en agosto de 2000) ha redondeado algunos de los grupos de características iniciados en versiones anteriores y agregado funcionalidades completamente nuevas, incluyendo soporte XML.

La versión que actualmente se está implementando está diseñada para aumentar la facilidad de uso del producto, facilidad de desarrollo de aplicaciones, robustez, rendimiento y dimensionabilidad.

NOTAS BIBLIOGRÁFICAS

Las diferencias entre las varias ediciones de SQL Server se describen en Delaney [2000] y también están disponibles en Web en www.microsoft.com/sql.

En www.microsoft.com/Downloads/Release.asp?ReleaseID=25503 está disponible información detallada sobre el uso del sistema certificado C2 con SQL Server.

El entorno de optimización de SQL Server está basado en el prototipo de optimizador Cascades, que Graefe [1995] propuso. Simmen et al. [1996] discute el esque-

ma para reducir las columnas de agrupación. Galindo-Legaria y Joshi [2001] presentan una serie de estrategias de ejecución que SQL Server considera durante la optimización basada en el coste. Información adicional sobre aspectos de autoajuste del servidor SQL se discuten en Chaudhuri et al. [1999]. Chaudhuri y Shim [1994] y Yan y Larson [1995] discuten la agregación parcial. Chatziantoniou y Ross [1997] y Galindo-Legaria y Joshi [2001] propusieron una alternativa utilizada por SQL Server para consultas SQL que requieren una autorreunión. Bajo este esquema el optimizador detecta el patrón y considera la ejecución por segmentos. Pellenkoft et al. [1997] discuten el esquema de optimización para generar el espacio completo, donde el optimizador utiliza transformaciones completas, locales y no redundantes.

Graefe et al. [1998] ofrece discusiones relacionadas con las operaciones de asociación que soportan agregación y reunión básica, con una serie de optimizaciones, extensiones y ajuste dinámico del sesgo de datos. Graefe et al. [1998] presentan la idea de reunir índices con el único propósito de ensamblar una fila con el conjunto de columnas necesarias en una consulta. Argumenta que esto algunas veces es más rápido que explorar una tabla base. Blakeley [1996] y Blakeley y Pizzo [2001] discuten respecto a la comunicación con el motor de almacenamiento a través de OLE-DB.

La Figura 27.11, que muestra un diagrama de bloques de arquitectura de alto nivel del soporte XML para SQL Server es de Rys [2001].



BIBLIOGRAFÍA

[Abbott y García-Molina 1999] R. Abbott y H. García-Molina, «Scheduling Real-Time Transactions: A Performance Evaluation», *ACM Transactions on Database Systems,* volumen 17, número 3 (1999), páginas 513-560.

[Abiteboul et al. 1995] S. Abiteboul, R. Hull y V. Vianu, *Foundations of Databases*, Addison Wesley (1995).

[Acharya et al. 1995] S. Acharya, R. Alonso, M. Franklin y S. Zdonik, «Broadcast Disks: Data Management for Asymmetric Communication Environments», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 19-210. También aparece en Imielinski y Korth [1996], Capítulo 12.

[Adali et al. 1996] S. Adali, K. S. Candan, Y. Papakonstantinou y V. S. Subrahmanian, «Query Caching and Optimization in Distributed Mediator Systems», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 137-148.

[Agarwal et al. 1996] S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan y S. Sarawagi, «On the computation of multidimensional attributes», *Proc. of the International Conf. on Very Large Databases*, Bombay, India (1996), páginas 506-521.

[Agrawal et al. 1992] R. Agrawal, S. P. Ghosh, T. Imielinski, B. R. Iyer y A. N. Swami, «An Interval Classifier for Database Mining Applications», *Proc. of the International Conf. on Very Large Databases* (1992), páginas 560-573.

[Agrawal et al. 1993] R. Agrawal, T. Imielinski y A. N. Swami, «Database Mining: A Performance Perspective», *IEEE Transactions on Knowledge and Data Engineering*, volumen 5, número 6 (1993), páginas 914-925.

[Agrawal et al. 2000] S. Agrawal, S. Chaudhuri y V. R. Narasayya, «Automated Selection of Materialized Views and Indexes in SQL Databases», *Proc. of the International Conf. on Very Large Databases* (2000), páginas 496-505.

[Agrawal y Gehani 1989] R. Agrawal y N. Gehani, «ODE (Object Database y Environment): The Language and the Data Model», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1989), páginas 36-45.

[Agrawal y Srikant 1994] R. Agrawal y R. Srikant, «Fast Algorithms for Mining Association Rules in Large Databases», *Proc. of the International Conf. on Very Large Databases* (1994), páginas 487-499.

[Aho et al. 1979a] A. V. Aho, C. Beeri y J. D. Ullman, «The Theory of Joins in Relational Databases», *ACM Transactions on Database Systems,* volumen 4, número 3 (1979), páginas 297-314.

[Aho et al. 1979b] V. Aho, Y. Sagiv y J. D. Ullman, «Equivalences among Relational Expressions», *SIAM Journal of Computing,* volumen 8, número 2 (1979), páginas 218-246.

[Aho et al. 1986] A. V. Aho, R. Sethi y J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison Wesley (1986).

[Aiken et al. 1995] A. Aiken, J. Hellerstein y J. Widom, «Static Analysis Techniques for Predicting the Behavior of Active Database Rules», *ACM Transactions on Database Systems*, volumen 20, número 1 (1995), páginas 3-41.

[Akl 1983] S. Akl, «Digital Signatures, A Tutorial Survey», *IEEE Computer,* volumen 16, número 2 (1983), páginas 15-24.

[Alonso y Korth 1993] R. Alonso y H. F. Korth, «Database System Issues in Nomadic Computing», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993), páginas 388-392.

[Anderson et al. 1992] D. P. Anderson, Y. Osawa y R. Govindan, «A File System for Continuous Media», *ACM Transactions on Database Systems,* volumen 10, número 4 (1992), páginas 311-337.

[Anderson et al. 1998] T. Anderson, Y. Breitbart, H. F. Korth y A. Wool, «Replication, Consistency and Practicality: Are These Mutually Exclusive?», *Proc. of the ACM SIGMOD Conf. on Management of Data*, Seattle,WA (1998).

[ANSI 1986] *American National Standard for Information Systems: Database Language SQL*. American National Standards Institute. FDT, ANSI X3,135-1986 (1986).

[ANSI 1989] *Database Language SQL With Integrity Enhancement, ANSI X3, 135-1989*. American National Standards Institute, Nueva York. También disponible como documento ISO/IEC 9075:1989 (1989).

[ANSI 1992] *Database Language SQL, ANSI X3,135-1992*. American National Standards Institute, Nueva York. También disponible como documento ISO/IEC 9075:1992 (1992).

[Antoshenkov 1995] G. Antoshenkov, «Bytealigned Bitmap Compression (poster abstract)», en *IEEE Data Compression Conf.* (1995).

[Apers et al. 1983] P. M. G. Apers, A. R. Hevner y S. B. Yao, «Optimization Algorithms for Distributed Queries», *IEEE Transactions on Software Engineering,* volumen SE-9, número 1 (1983), páginas 57-68.

[Apt y Pugin 1987] K. R. Apt y J. M. Pugin, «Maintenance of Stratified Database Viewed as a Belief Revision System», *Proc. of the ACM Symposium on Principles of Database Systems* (1987), páginas 136-145.

[Aref et al. 1995a] W. Aref, D. Barbará, D. Lopresti y A. Tomkins, *Ink as a First-Class Multimedia Object*, Jajodia y Subramanian (1995).

[Aref et al. 1995b] W. Aref, D. Barbará y P. Vallabhaneni, «The Handwritten Trie: Indexing Electronic Ink», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 151-162.

[Armstrong 1974] W. W. Armstrong, «Dependency Structures of Data Base Relationships», en *Proc. of the 1974 IFIP Congress* (1974), páginas 580-583.

[Astrahan et al. 1976] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. G. y P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade y V. Watson, «System R, A Relational Approach to Data Base Management», *ACM Transactions on Database Systems*, volumen 1, número 2 (1976), páginas 97-137.

[**Astrahan et al. 1979**] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, J. N. Gray, W. F. King, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, G. R. Putzolu,M. Schkolnick, P. G. Selinger, D. R. Slutz, H. R. Strong, P. Tiberio, I. L. Traiger, B.W. Wade y R. A. Yost, «System R: A Relational Database Management System», *IEEE Computer,* volumen 12, número 5 (1979), páginas 43-48.

[**Attar et al. 1984**] R. Attar, P. A. Bernstein y N. Goodman, «Site Initialization, Recovery and Backup in a Distributed Database System», *IEEE Transactions on Software Engineering*, volumen SE-10, número 6 (1984), páginas 645-650.

[**Atzeni y Antonellis 1993**] P. Atzeni y V. D. Antonellis, *Relational Database Theory*, Benjamin Cummings, Menlo Park (1993).

[**Badal y Popek 1979**] D. S. Badal y G. Popek, «Cost and Performance Analysis of Semantic Integrity Validation Methods», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1979), páginas 109-115.

[**Badrinath y Ramamritham 1992**] B. R. Badrinath y K. Ramamritham, «Semantics-Based Concurrency Control: Beyond Commutativity», *ACM Transactions on Database Systems,* volumen 17, número 1 (1992), páginas 163-199.

[**Baeza-Yates y Ribeiro-Neto 1999**] R. Baeza-Yates y B. Ribeiro-Neto, *Modern Information Retrieval*, Addison Wesley (1999).

[**Bamford et al. 1998**] R. Bamford, D. Butler, B. Klots y N. MacNaughton, «Architecture of Oracle Parallel Server», *Proc. of the International Conf. on Very Large Databases* (1998), páginas 669-670.

[**Bancilhon et al. 1989**] F. Bancilhon, S. Cluet y C. Delobel, «A Query Language for the O_2 Object-Oriented Database», *Proc. of the Second Workshop on Database Programming Languages* (1989).

[**Bancilhon y Buneman 1990**] F. Bancilhon y P. Buneman, *Advances in Database Programming Languages*, ACM Press, Nueva York (1990).

[**Bancilhon y Ramakrishnan 1986**] F. Bancilhon y R. Ramakrishnan, «An Amateur's Introduction to Recursive Query-Processing Strategies», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1986), páginas 16-52.

[**Bancilhon y Spyratos 1981**] F. Bancilhon y N. Spyratos, «Update Semantics and Relational Views», *ACM Transactions on Database Systems*, volumen 6, número 4 (1981), páginas 557-575.

[**Banerjee et al. 1987**] J. Banerjee, W. Kim, H. J. Kim y H. F. Korth, «Semantics and Implementation of Schema Evolution in Object-Oriented Databases», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987), páginas 311-322.

[**Banerjee et al. 2000**] S. Banerjee, V. Krishnamurthy, M. Krishnaprasad y R. Murthy, «Oracle8i—The XML Enabled Data Management System», *Proc. of the International Conf. on Data Engineering* (2000), páginas 561-568.

[**Barbará e Imielinski 1994**] D. Barbará y T. Imielinski, «Sleepers and Workaholics: Caching Strategies in Mobile Environments», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1994), páginas 1-12.

[**Barbará et al. 1992**] D. Barbará, H. García-Molina y D. Porter, «TheManagement of Probabilistic Data», *IEEE Transactions on Knowledge and Data Engineering*, volumen 4, número 5 (1992), páginas 487-502.

[**Barclay et al. 1982**] D. K. Barclay, E. R. Byrne y F. K. Ng, «A Real-Time Database Management System for No.5 ESS», *Bell System Technical Journal,* volumen 61, número 9 (1982), páginas 2423-2437.

[**Baru et al. 1995**] C. Baru et al., «DB2 Parallel Edition», *IBM Systems Journal,* volumen 34, número 2 (1995), páginas 292-322.

[**Baru et al. 1999**] C. K. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov y V. Chu, «XML-Based Information Mediation with MIX», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1999), páginas 597-599.

[**Bassiouni 1988**] M. Bassiouni, «Single-site and Distributed Optimistic Protocols for Concurrency Control», *IEEE Transactions on Software Engineering,* volumen SE-14, número 8 (1988), páginas 1071-1080.

[**Batini et al. 1992**] C. Batini, S. Ceri y S. Navathe, *Database Design: An Entity-Relationship Approach*, Benjamin Cummings, Redwood City (1992).

[**Bayer 1972**] R. Bayer, «Symmetric Binary B-trees: Data Structure and Maintenance Algorithms», *Acta Informatica,* volumen 1, número 4 (1972), páginas 290-306.

[**Bayer et al. 1978**] R. Bayer, R. M. Graham y G. Seegmuller, editores, *Operating Systems: An Advanced Course*, Springer Verlag (1978).

[**Bayer et al. 1980**] R. Bayer, H. Heller y A. Reiser, «Parallelism and Recovery in Database Systems», *ACM Transactions on Database Systems,* volumen 5, número 2 (1980), páginas 139-156.

[**Bayer y McCreight 1972**] R. Bayer y E. M. McCreight, «Organization and Maintenance of Large Ordered Indices», *Acta Informatica,* volumen 1, número 3 (1972), páginas 173-189.

[**Bayer y Schkolnick 1977**] R. Bayer y M. Schkolnick, «Concurrency of Operating on B-trees», *Acta Informatica*, volumen 9, número 1 (1977), páginas 1-21.

[**Bayer y Unterauer 1977**] R. Bayer y K. Unterauer, «Prefix B-trees», *ACM Transactions on Database Systems*, volumen 2, número 1 (1977), páginas 11-26.

[**Beckmann et al. 1990**] N. Beckmann, H. P. Kriegel, R. Sc̈hneider y B. Seeger, «The R-tree: An Efficient and Robust Access Method for Points and Rectangles», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990), páginas 322-331.

[**Beech 1988**] D. Beech, «OSQL: A Language forMigrating from SQL to Object Databases», *Proc. of the International Conf. on Extending Database Technology* (1988).

[**Beeri y Ramakrishnan 1991**] C. Beeri y R. Ramakrishnan, «On the Power of Magic», *Journal of Logic Programming*, volumen 10, números 3 y 4 (1991), páginas 255-300.

[**Beeri et al. 1977**] C. Beeri, R. Fagin y J. H. Howard, «A Complete Axiomatization for Functional and Multivalued Dependencies», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1977), páginas 47-61.

[**Beeri et al. 1989**] C. Beeri, P. A. Bernstein y N. Goodman, «A Model for Concurrency in Nested Transactions Systems», *Journal of the ACM,* volumen 36, número 2 (1989).

[**Bell y Grimson 1992**] D. Bell y J. Grimson, *Distributed Database Systems*, Addison Wesley (1992).

[**Bell y LaPadula 1976**] D. E. Bell y L. J. LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation*, Mitre Corporation, Bedford (1976).

[Bello et al. 1998] R. G. Bello, K. Dias, A. Downing, J. Feenan, J. Finnerty, W. D. Norcott, H. Sun, A. Witkowski y M. Ziauddin, «Materialized Views in Oracle», *Proc. of the International Conf. on Very Large Databases* (1998), páginas 659-664.

[Benneworth et al. 1981] R. L. Benneworth, C. D. Bishop, C. J. M. Turnbull, W. D. Holman y F. M. Monette, «The Implementation of GERM, an Entity-Relationship Data Base Management System», *Proc. of the International Conf. on Very Large Databases* (1981), páginas 478-484.

[Bentley 1975] J. L. Bentley, «Multidimensional Binary Search Trees Used for Associative Searching», *Communications of the ACM,* volumen 18, número 9 (1975), páginas 509-517.

[Berenson et al. 1995] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil y P. O'Neil, «A Critique of ANSI SQL Isolation Levels», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 1-10.

[Bernstein et al. 1978] P. A. Bernstein, N. Goodman, J. B. Rothnie y C. H. Papadimitriou, «Analysis of Serializability of SDD-1: A System of Distributed Databases (the Fully Redundant Case)», *IEEE Transactions on Software Engineering*, volumen SE-4, número 3 (1978), páginas 154-168.

[Bernstein et al. 1980a] P. A. Bernstein, B. Blaustein y E. Clarke, «Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data», *Proc. of the International Conf. on Very Large Databases* (1980).

[Bernstein et al. 1980b] P. A. Bernstein, D. W. Shipman y J. B. Rothnie, «Concurrency Control in a System for Distributed Databases (SDD-1)», *ACM Transactions on Database Systems*, volumen 5, número 1 (1980), páginas 18-51.

[Bernstein et al. 1983] P. A. Bernstein, N. Goodman y M. Y. Lai, «Analyzing Concurrency Control when User and System Operations Differ», *IEEE Transactions on Software Engineering*, volumen SE-9, número 3 (1983), páginas 233-239.

[Bernstein et al. 1987] A. Bernstein, V. Hadzilacos y N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison Wesley (1987).

[Bernstein et al. 1998] P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. García-Molina, J. Gray, J. Held, J. Hellerstein, H. V. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker y J. Ullman, «The Asilomar Report on Database Research», *ACM SIGMOD Record,* volumen 27, número 4 (1998).

[Bernstein y Chiu 1981] P. A. Bernstein y D.W. Chiu, «Using Semijoins to Solve Relational Queries», *Journal of the ACM,* volumen 28, número 1 (1981), páginas 25-40.

[Bernstein y Goodman 1980] P. A. Bernstein y N. Goodman, «Timestamp-based Algorithms for Concurrency Control in Distributed Database Systems», *Proc. of the International Conf. on Very Large Databases* (1980), páginas 285-300.

[Bernstein y Goodman 1981a] P. A. Bernstein y N. Goodman, «Concurrency Control in Distributed Database Systems», *ACM Computing Survey,* volumen 13, número 2 (1981), páginas 185-221.

[Bernstein y Goodman 1981b] P. A. Bernstein y N. Goodman, «The Power of Natural Semijoins», *SIAM Journal of Computing*, volumen 10, número 4 (1981), páginas 751-771.

[Bernstein y Goodman 1982] P. A. Bernstein y N. Goodman, «A Sophisticate's Introduction to Distributed Database Concurrency Control», *Proc. of the International Conf. on Very Large Databases* (1982), páginas 62-76.

[Bernstein y Newcomer 1997] P. A. Bernstein y E. Newcomer, *Principles of Transaction Processing*, Morgan Kaufmann (1997).

[Berson et al. 1995] S. Berson, L. Golubchik y R. R. Muntz, «Fault Tolerant Design of Multimedia Servers», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 364-375.

[Bertino y Kim 1989] E. Bertino y W. Kim, «Indexing Techniques for Queries on Nested Objects», *IEEE Transactions on Knowledge and Data Engineering* (1989).

[Bharat y Henzinger 1998] K. Bharat y M. R. Henzinger, «Improved Algorithms for Topic Distillation in a Hyperlinked Environment», *Proc. of the ACM SIGIR Conf. on Research And Development in Information Retrieval* (1998), páginas 104-111.

[Biliris y Orenstein 1994] A. Biliris y J. Orenstein, «Object Storage Management Architectures», en *Dogac et al. [1994]*, páginas 185-200 (1994).

[Biskup et al. 1979] J. Biskup, U. Dayal y P. A. Bernstein, «Synthesizing Independent Database Schemas», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1979), páginas 143-152.

[Bitton et al. 1983] D. Bitton, D. J. DeWitt y C. Turbyfill, «Benchmarking Database Systems: A Systematic Approach», *Proc. of the International Conf. on Very Large Databases* (1983).

[Bitton y Gray 1988] D. Bitton y J. N. Gray, «Disk Shadowing», *Proc. of the International Conf. on Very Large Databases* (1988), páginas 331-338.

[Bjork 1973] L. A. Bjork, «Recovery Scenario for a DB/DC System», *Proc. of the ACM Annual Conf.* (1973), páginas 142-146.

[Blakeley 1996] J. A. Blakeley, «Data Access for the Masses through OLE DB», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 161-172.

[Blakeley et al. 1986] J. A. Blakeley, P. Larson y F. W. Tompa, «Efficiently Updating Materialized Views», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1986), páginas 61-71.

[Blakeley et al. 1989] J. Blakeley, N. Coburn y P. Larson, «Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates», *ACM Transactions on Database Systems*, volumen 14, número 3 (1989), páginas 369-400.

[Blakeley et al. 1993] J. A. Blakeley, W. J. McKenna y G. Graefe, «Experiences Building the Open OODB Query Optimizer», *Proc. of the ACM SIGMOD Conf. on Management of Data*, Washington, DC. (1993), páginas 287-295.

[Blakeley y Pizzo 2001] J. A. Blakeley y M. Pizzo, «Enabling Component Databases with OLE DB», en K. R. Dittrich y A. Geppert, editores, *Component Database Systems*, Morgan Kauffmann Publishers (2001), páginas 139-173.

[Blasgen y Eswaran 1976] M. W. Blasgen y K. P. Eswaran, «On the Evaluation of Queries in a Relational Database System», *IBM Systems Journal,* volumen 16, (1976), páginas 363-377.

[Boral et al. 1990] H. Boral,W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith y P. Val-

duriez, «Prototyping Bubba, a Highly Parallel Database System», *IEEE Transactions on Knowledge and Data Engineering*, volumen 2, número 1 (1990).

[Boyce et al. 1975] R. Boyce, D. D. Chamberlin, W. F. King y M. Hammer, «Specifying Queries as Relational Expressions», *Communications of the ACM,* volumen 18, número 11 (1975), páginas 621-628.

[Breese et al. 1998] J. Breese, D. Heckerman y C. Kadie, «Empirical Analysis of Predictive Algorithms for Collaborative Filtering», *Proc. of the Conf. on Uncertainty in Artificial Intelligence*, Morgan Kaufmann (1998).

[Breitbart et al. 1990] Y. Breitbart, A. Silberschatz y G. Thompson, «Reliable Transaction Management in a Multidatabase System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990).

[Breitbart et al. 1991] Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz y A. Silberschatz, «On Rigorous Transaction Scheduling», *IEEE Transactions on Software Engineering*, volumen SE-17, número 9 (1991), páginas 954-961.

[Breitbart et al. 1992] Y. Breitbart, A. Silberschatz y G. Thompson, «TransactionManagement Issues in a Failure-prone Multidatabase System Environment», *VLDB Journal*, volumen 1, número 1 (1992), páginas 1-39.

[Breitbart et al. 1999a] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri y A. Silberschatz, «Update Propagation Protocols For Replicated Databases», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1999), páginas 97-108.

[Breitbart et al. 1999b] Y. Breitbart, H. Korth, A. Silberschatz y S. Sudarshan. «Distributed Databases», en *Encyclopedia of Electrical and Electronics Engineering*. John Wiley and Sons (1999).

[Bridge et al. 1997] W. Bridge, A. Joshi, M. Keihl, T. Lahiri, J. Loaiza y N. MacNaughton, «The Oracle Universal Server Buffer», *Proc. of the International Conf. on Very Large Databases* (1997), páginas 590-594.

[Brin y Page 1998] S. Brin y L. Page, «The Anatomy of a Large-Scale Hypertextual Web Search Engine», *Proc. of the International World Wide Web Conf.* (1998).

[Brinkhoff et al. 1993] T. Brinkhoff, H.-P. Kriegel y B. Seeger, «Efficient Processing of Spatial Joins Using R-trees», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993), páginas 237-246.

[Brown et al. 1994] K. P. Brown, M. Mehta, M. Carey y M. Livny, «Towards Automated Performance Tuning for Complex Overloads», *Proc. of the International Conf. on Very Large Databases* (1994).

[Buckley y Silberschatz 1983] G. Buckley y A. Silberschatz, «Obtaining Progressive Protocols for a Simple Multiversion Database Model», *Proc. of the International Conf. on Very Large Databases* (1983), páginas 74-81.

[Buckley y Silberschatz 1984] G. Buckley y A. Silberschatz, «Concurrency Control in Graph Protocols by Using Edge Locks», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1984), páginas 45-50.

[Buckley y Silberschatz 1985] G. Buckley y A. Silberschatz, «Beyond Two-Phase Locking», *Journal of the ACM*, volumen 32, número 2 (1985), páginas 314-326.

[Bulmer 1979] M. G. Bulmer, *Principles of Statistics*, Dover Publications (1979).

[Burkhard 1976] W. A. Burkhard, «Hashing and Trie Algorithms for Partial Match Retrieval», *ACM Transactions on Database Systems,* volumen 1, número 2 (1976), páginas 175-187.

[Burkhard 1979] W. A. Burkhard, «Partial-match Hash Coding: Benefits of Redundancy», *ACM Transactions on Database Systems*, volumen 4, número 2 (1979), páginas 228-239.

[Cannan y Otten 1993] S. Cannan y G. Otten, *SQL—The Standard Handbook*, McGraw Hill (1993).

[Carey 1983] M. J. Carey, «Granularity Hierarchies in Concurrency Control», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1983), páginas 156-165.

[Carey et al. 1986] M. J. Carey, D. DeWitt, J. Richardson y E. Shekita, «Object and File Management in the EXODUS Extensible Database System», *Proc. of the International Conf. on Very Large Databases* (1986), páginas 91-100.

[Carey et al. 1990] M. J. Carey, D. DeWitt, G. Graefe, D. Haight, D. S. J. Richardson, E. Shekita y S. Vandenberg. «The EXODUS Extensible DBMS Project: An Overview», en *Zdonik y Maier [1990]*, páginas 474-499 (1990).

[Carey et al. 1991] M. Carey, M. Franklin, M. Livny y E. Shekita, «Data Caching Tradeoffs in Client-Server DBMS Architectures», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1991).

[Carey et al. 1993] M. J. Carey, D. DeWitt y J. Naughton, «The OO7 Benchmark», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993).

[Carey et al. 1999] M. J. Carey, D. D. Chamberlin, S. Narayanan, B. Vance, D. Doole, S. Rielau, R. Swagerman y N. Mattos, «O-O, What have they done to DB2?», en *Proc. of the International Conf. on Very Large Databases* (1999), páginas 542-553.

[Carey et al. 2000] M. J. Carey, J. Kiernan, J. Shanmugasundaram, E. J. Shekita y S. N. Subramanian, «XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents», *Proc. of the International Conf. on Very Large Databases* (2000), páginas 646-648.

[Cattell 2000] R. Cattell, editor, *The Object Database Standard: ODMG 3.0*, Morgan Kaufmann (2000).

[Cattell y Skeen 1992] R. Cattell y J. Skeen, «Object Operations Benchmark», *ACM Transactions on Database Systems*, volumen 17, número 1 (1992).

[Ceri y Owicki 1983] S. Ceri y S. Owicki, «On the Use of Optimistic Methods for Concurrency Control in Distributed Databases», *Proc. of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks* (1983).

[Ceri y Pelagatti 1983] S. Ceri y G. Pelagatti, «Correctness of Query Execution Strategies in Distributed Databases», *ACM Transactions on Database Systems,* volumen 8, número 4 (1983), páginas 577-607.

[Ceri y Pelagatti 1984] S. Ceri y G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw Hill (1984).

[Chakrabarti 1999] S. Chakrabarti, «Recent results in automatic Web resource discovery», *ACM Computing Surveys*, volumen 31, número 4es (1999).

[Chakrabarti 2000] S. Chakrabarti, «Data mining for hypertext: A tutorial survey», *SIGKDD Explorations*, volumen 1, número 2 (2000), páginas 1-11.

[Chakrabarti et al. 1998] S. Chakrabarti, S. Sarawagi y B. Dom, «Mining Surprising Patterns Using Temporal Description Length», *Proc. of the International Conf. on Very Large Databases* (1998), páginas 606-617.

[Chakravarthy et al. 1990] U. S. Chakravarthy, J. Grant y J. Minker, «Logic-Based Approach to Semantic Query Optimization», *ACM Transactions on Database Systems,* volumen 15, número 2 (1990), páginas 162-207.

[Chamberlin 1996] D. Chamberlin, *Using the new DB2: IBM's object-relational database system*, Morgan Kaufmann (1996).

[Chamberlin 1998] D. D. Chamberlin, *A complete guide to DB2 Universal Database*, Morgan Kaufmann (1998).

[Chamberlin et al. 1976] D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner y B. W. Wade, «SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control», *IBM Journal of Research and Development*, volumen 20, número 6 (1976), páginas 560-575.

[Chamberlin et al. 1981] D. D. Chamberlin, M. M. Astrahan, M. W. Blasgen, J. N. Gray, W. F. King, B. G. Lindsay, R. A. Lorie, J.W. Mehl, T. G. Price, P. G. Selinger, M. Schkolnick, D. R. Slutz, I. L. Traiger, B. W. Wade y R. A. Yost, «A History and Evaluation of System R», *Communications of the ACM,* volumen 24, número 10 (1981), páginas 632-646.

[Chamberlin et al. 2000] D. D. Chamberlin, J. Robie y D. Florescu, «Quilt: An XML Query Language for Heterogeneous Data Sources», *Proc. of the International Workshop on the Web and Databases (WebDB)* (2000), páginas 53-62.

[Chamberlin y Boyce 1974] D. D. Chamberlin y R. F. Boyce, «SEQUEL: A Structured English Query Language», *ACM SIGMOD Workshop on Data Description, Access, and Control* (1974), páginas 249-264.

[Chan e Ioannidis 1998] C.-Y. Chan y Y. E. Ioannidis, «Bitmap Index Design and Evaluation», en *Proc. of the ACM SIGMOD Conf. on Management of Data* (1998).

[Chan e Ioannidis 1999] C.-Y. Chan y Y. E. Ioannidis, «An Efficient Bitmap Encoding Scheme for Selection Queries», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1999).

[Chandra y Harel 1982] A. K. Chandra y D. Harel, «Structure and Complexity of Relational Queries», *Journal of Computer and System Sciences,* volumen 15, número 10 (1982), páginas 99-128.

[Chandy et al. 1975] K. M. Chandy, J. C. Browne, C. W. Dissley y W. R. Uhrig, «Analytic Models for Rollback and Recovery Strategies in Database Systems», *IEEE Transactions on Software Engineering*, volumen SE-1, número 1 (1975), páginas 100-110.

[Chandy et al. 1983] K. M. Chandy, L. M. Haas y J. Misra, «Distributed Deadlock Detection», *ACM Transactions on Database Systems,* volumen 1, número 2 (1983), páginas 144-156.

[Chandy y Misra 1982] K. M. Chandy y J. Misra, «A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1982), páginas 157-164.

[Chatziantoniou y Ross 1997] D. Chatziantoniou y K. A. Ross, «Groupwise Processing of Relational Queries», *Proc. of the International Conf. on Very Large Databases* (1997), páginas 476-485.

[Chaudhuri et al. 1995] S. Chaudhuri, R. Krishnamurthy, S. Potamianos y K. Shim, «Optimizing Queries with Materialized Views», *Proc. of the International Conf. on Data Engineering*, Taipei, Taiwan (1995).

[Chaudhuri et al. 1999] S. Chaudhuri, E. Christensen, G. Graefe, V. Narasayya y M. Zwilling, «Self Tuning Technology in Microsoft SQL Server», *IEEE Data Engineering Bulletin*, volumen 22, número 2 (1999).

[Chaudhuri y Narasayya 1997] S. Chaudhuri y V. Narasayya, «An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server», *Proc. of the International Conf. on Very Large Databases* (1997).

[Chaudhuri y Shim 1994] S. Chaudhuri y K. Shim, «Including Group-By in Query Optimization», en *Proc. of the International Conf. on Very Large Databases* (1994).

[Chawathe 1999] S. S. Chawathe, «Describing and Manipulating XML Data», *IEEE Data Engineering Bulletin (Special Issue on XML)* (1999), páginas 3-9.

[Chen 1976] P. P. Chen, «The Entity-Relationship Model: Toward a Unified View of Data», *ACM Transactions on Database Systems,* volumen 1, número 1 (1976), páginas 9-36.

[Chen et al. 1994] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz y D. A. Patterson, «RAID: High-Performance, Reliable Secondary Storage», *ACM Computing Survey*, volumen 26, número 2 (1994).

[Chen et al. 1995] M.-S. Chen, H.-I. Hsiao, C.-S. Li y P. S. Yu, «Using Rotational Mirrored Declustering for Replica Placement in a Disk Array Based Video Server», *Proc. of the ACM International Multimedia Conf.* (1995), páginas 121-130.

[Chen y Patterson 1990] P. Chen y D. Patterson, «Maximizing Performance in a Striped Disk Array», *Proc. of the Seventeenth Annual International Symposium on Computer Architecture* (1990).

[Cheng y Xu 2000] J. M. Cheng y J. Xu, «XML and DB2», *Proc. of the International Conf. on Data Engineering* (2000), páginas 569-573.

[Chervenak et al. 1995] A. L. Chervenak, D. A. Patterson y R. H. Katz, «Choosing the Best Storage System for Video Service», *Proc. of the ACM International Multimedia Conf.* (1995), páginas 109-120.

[Chiu y Ho 1980] D. M. Chiu y Y. C. Ho, «A Methodology for Interpreting Tree Queries into Optimal Semi-join Expressions», *Proc. of theACM SIGMOD Conf. on Management of Data* (1980), páginas 169-178.

[Chomicki 1992] J. Chomicki, «History-less checking of Dynamic Integrity Constraints», *Proc. of the International Conf. on Data Engineering* (1992).

[Chomicki 1995] J. Chomicki, «Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding», *ACM Transactions on Database Systems,* volumen 20, número 2 (1995), páginas 149-186.

[Chou et al. 1985] H. T. Chou, D. J. Dewitt, R. H. Katz y A. C. Klug, «The Wisconsin Storage System Software», *Software—Practice and Experience,* volumen 15, número 10 (1985), páginas 943-962.

[Chou y Dewitt 1985] H. T. Chou y D. J. Dewitt, «An Evaluation of Buffer Management Strategies for Relational Database Systems», *Proc. of the International Conf. on Very Large Databases* (1985), páginas 127-141.

[Chrysanthis y Ramamritham 1994] P. Chrysanthis y K. Ramamritham, «Synthesis of Extended Transaction Models Using ACTA», *ACM Transactions on Database Systems,* volumen 19, número 3 (1994), páginas 450-491.

[Clifford et al. 1994] J. Clifford, A. Croker y A. Tuzhilin, «On Completeness of Historical Relational Query Lan-

guages», *ACM Transactions on Database Systems,* volumen 19, número 1 (1994), páginas 64-116.

[Clifford y Tansel 1985] J. Clifford y A. Tansel, «On an Algebra for Historical Relational Databases: Two Views», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1985), páginas 247-267.

[Cochrane et al. 1996] R. Cochrane, H. Pirahesh y N. M. Mattos, «Integrating Triggers and Declarative Constraints in SQL Database Sytems», *Proc. of the International Conf. on Very Large Databases* (1996).

[Codd 1970] E. F. Codd, «A Relational Model for Large Shared Data Banks», *Communications of the ACM*, volumen 13, número 6 (1970), páginas 377-387.

[Codd 1972] E. F. Codd. «Further Normalization of the Data Base Relational Model», en *Rustin [1972]*, páginas 33-64 (1972).

[Codd 1979] E. F. Codd, «Extending the Database Relational Model to Capture More Meaning», *ACM Transactions on Database Systems,* volumen 4, número 4 (1979), páginas 397-434.

[Codd 1982] E. F. Codd, «The 1981 ACM Turing Award Lecture: Relational Database: A Practical Foundation for Productivity», *Communications of the ACM,* volumen 25, número 2 (1982), páginas 109-117.

[Codd 1990] E. F. Codd, *The Relational Model for Database Management: Version 2*, Addison Wesley (1990).

[Comer 1979] D. Comer, «The Ubiquitous B-tree», *ACM Computing Survey,* volumen 11, número 2 (1979), páginas 121-137.

[Comer y Droms 1999] D. E. Comer y R. E. Droms, *Computer Networks and Internets*, 2.ª edición, Prentice Hall (1999).

[Cook 1996] M. A. Cook, *Building Enterprise Information Architecture: Reengineering Information Systems*, Prentice Hall (1996).

[Cook et al. 1999] J. Cook, R. Harbus y T. Shirai, *The DB2 Universal Database V6.1 certification guide: For UNIX, Windows, and OS/2*, Prentice Hall (1999).

[Cormen et al. 1990] T. Cormen, C. Leiserson y R. Rivest, *Introduction to Algorithms*, MIT Press (1990).

[Cosmadakis y Papadimitriou 1984] S. S. Cosmadakis y C. H. Papadimitriou, «Updates of Relational Views», *Journal of the ACM*, volumen 31, número 4 (1984), páginas 742-760.

[Daemen y Rijmen 2000] J. Daemen y V. Rijmen. «The Block Cipher Rijndael», en J.-J. Quisquater y B. Schneier, editores, *Smart Card Research y Applications (LNCS 1820)*, páginas 288-296. Springer Verlag (2000).

[Dalvi et al. 2001] N. N. Dalvi, S. K. Sanghai, P. Roy y S. Sudarshan, «Pipelining in Multi-Query Optimization», *Proc. of the ACM Symposium on Principles of Database Systems* (2001).

[Daniels et al. 1982] D. Daniels, P. G. Selinger, L. M. Haas, B. G. Lindsay, C. Mohan, A. Walker y P. F. Wilms. «An Introduction to Distributed Query Compilation in R*», en *Schneider [1982]* (1982).

[Dar et al. 1996] S. Dar, H. V. Jagadish, A. Levy y D. Srivastava, «Answering Queries with Aggregation Using Views», *Proc. of the International Conf. on Very Large Databases* (1996).

[Date 1986] C. J. Date, *Relational Databases: selected Writings*, Addison Wesley (1986).

[Date 1989] C. Date, *A guide to DB2*, Addison Wesley (1989).

[Date 1990] C. J. Date, *Relational Database Writings, 1985-1989*, Addison Wesley (1990).

[Date 1993a] C. J. Date, «How SQL Missed the Boat», *Database Programming and Design*, volumen 6, número 9 (1993).

[Date 1993b] C. J. Date, «The Outer Join», en *IOCOD-2,* John Wiley and Sons (1993), páginas 76-106.

[Date 1995] C. J. Date, *An Introduction to Database Systems*, 6.ª edición, Addison Wesley (1995).

[Date y Darwen 1997] C. J. Date y G. Darwen, *A Guide to the SQL Standard*, 4.ª edición, Addison Wesley (1997).

[Davies 1973] C. T. Davies, «Recovery Semantics for a DB/DC System», *Proc. of the ACM Annual Conference* (1973), páginas 136-141.

[Davis et al. 1983] C. Davis, S. Jajodia, P. A. Ng y R. Yeh, editores, *Entity-Relationship Approach to Software Engineering*, North Holland (1983).

[Davison y Graefe 1994] D. L. Davison y G. Graefe, «Memory-Contention Responsive Hash Joins», *Proc. of the International Conf. on Very Large Databases* (1994).

[Dayal 1987] U. Dayal, «Of Nests and Trees: A Unified Approach to Processing Queries that Contain Nested Subqueries, Aggregates and Quantifiers», *Proc. of the International Conf. on Very Large Databases* (1987), páginas 197-208.

[Dayal y Bernstein 1978] U. Dayal y P. A. Bernstein, «The Updatability of Relational Views», en *Proc. of the International Conf. on Very Large Databases* (1978), páginas 368-377.

[Dayal et al. 1982] U. Dayal, N. Goodman y R. H. Katz, «An Extended Relational Algebra with Control over Duplicate Elimination», *Proc. of the ACM Symposium on Principles of Database Systems* (1982).

[Dayal et al. 1990] U. Dayal, M. Hsu y R. Ladin, «Organizing Long-Running Activities with Triggers and Transactions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990), páginas 204-215.

[Dayal et al. 1991] U. Dayal, M. Hsu y R. Ladin, «A Transactional Model for Long-Running Activities», *Proc. of the International Conf. on Very Large Databases* (1991), páginas 113-122.

[de Prycker 1993] M. de Prycker, *Asynchronous Transfer Mode: Solution for Broadband ISDN*, 2.ª edición, Ellis Horwood (1993).

[Delaney 2000] K. Delaney, *Inside Microsoft SQL Server 2000*, Microsoft Press (2000).

[Denning y Denning 1979] D. E. Denning y P. J. Denning, «Data Security», *ACM Computing Survey*, volumen 11, número 3 (1979), páginas 227-250.

[Derr et al. 1993] M. A. Derr, S. Morishita y G. Phipps, «Design and Implementation of the Glue-Nail Database System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993), páginas 147-156.

[Deshpande y Larson 1992] V. Deshpande y P. A. Larson, «The Design and Implementation of a Parallel Join Algorithm for Nested Relations on Shared-MemoryMultiprocessors», *Proc. of the International Conf. on Data Engineering* (1992).

[Deutsch et al. 1999a] A. Deutsch, M. Fernández, D. Florescu, D. M. Alon Levy y D. Suciu, «Querying XML Data», *IEEE Data Engineering Bulletin (Special Issue on XML)* (1999), páginas 10-18.

[Deutsch et al. 1999b] A. Deutsch, M. Fernández, D. Florescu, A. Levy y D. Suciu, «A Query Language for XML»,

Proc. of the International World Wide Web Conf. (1999). (XML-QL enviado también el consorcio World Wide Web Consortium http://www.w3.org/TR/1998/NOTExml-ql-19980819).

[Deux 1991] O. Deux, «The *O*2 System», *Communications of the ACM,* volumen 34, número 10 (1991), páginas 34-49.

[DeWitt 1990] D. DeWitt, «The Gamma Database Machine Project», *IEEE Transactions on Knowledge and Data Engineering*, volumen 2, número 1 (1990).

[DeWitt et al. 1984] D. DeWitt, R. Katz, F. Olken, L. Shapiro, M. Stonebraker y D. Wood, «Implementation Techniques for Main Memory Databases», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1984), páginas 1-8.

[DeWitt et al. 1986] D. DeWitt, R. Gerber, G. Graefe, M. Heytens, K. Kumar y M. Muralikrishna, «A High Performance Dataflow Database Machine», *Proc. of the International Conf. on Very Large Databases* (1986).

[DeWitt et al. 1992] D. DeWitt, J. Naughton, D. Schneider y S. Seshadri, «Practical Skew Handling in Parallel Joins», *Proc. of the International Conf. on Very Large Databases* (1992).

[DeWitt y Gray 1992] D. DeWitt y J. Gray, «Parallel Database Systems: The Future of High Performance Database Systems», *Communications of the ACM,* volumen 35, número 6 (1992), páginas 85-98.

[Dias et al. 1989] D. Dias, B. Iyer, J. Robinson y P. Yu, «Integrated Concurrency-Coherency Controls for Multisystem Data Sharing», *Software Engineering*, volumen 15, número 4 (1989), páginas 437-448.

[Diffie y Hellman 1979] W. Diffie y M. E. Hellman, «Privacy and Authentication», *Proc. of the IEEE*, volumen 67, número 3 (1979), páginas 397-427.

[Dijkstra 1965] E. W. Dijkstra, «Cooperating Sequential Processes», informe técnico, Technological University, Eindhoven, Países Bajos, EWD-123 (1965).

[Dippert y Levy 1993] B. Dippert y M. Levy, *Designing with FLASH Memory*, Annabooks (1993).

[Dogac et al. 1994] A. Dogac, M. T. Ozsu, A. Biliris y T. Selis, *Advances in Object-Oriented Database Systems*, volume 130, Springer Verlag, Computer and Systems Sciences, NATO ASI Series F (1994).

[Douglis et al. 1994] F. Douglis, R. Cáceres, B. Marsh, F. Kaashoek, K. Li y J. Tauber, «Storage Alternatives for Mobile Computers», *Symposium on Operating Systems Design and Implementation* (1994), páginas 25-37. Versión actualizada en Douglis et al. [1996].

[Douglis et al. 1996] F. Douglis, R. Caceres, M. F. Kaashoek, P. Krishnan, K. Li, B. Marsh y J. Tauber. «Storage Alternatives for Mobile Computers», en *Imielinski and Korth [1996], Capítulo 18* (1996).

[Draper et al. 2001] D. Draper, A. Y. Halevy y D. S. Weld, «The Nimble XML Data Integration system», *Proc. of the International Conf. on Data Engineering* (2001), páginas 155-160.

[Du y Elmagarmid 1989] W. Du y A. Elmagarmid, «Quasi Serializability: A Correctness Criterion for Global Database Consistency in InterBase», *Proc. of the International Conf. on Very Large Databases* (1989), páginas 347-356.

[Dubois y Thakkar 1992] M. Dubois y S. Thakkar, editores, *Scalable Shared Memory Multiprocessors*, Kluwer Academic Publishers (1992).

[Duncan 1990] R. Duncan, «A Survey of Parallel Computer Architectures», *IEEE Computer*, volumen 23, número 2 (1990), páginas 5-16.

[Eisenberg andMelton 1999] A. Eisenberg y J. Melton, «SQL:1999, formerly known as SQL3», *ACM SIGMOD Record*, volumen 28, número 1 (1999).

[Ellis 1980a] C. S. Ellis, «Concurrent Search and Insertion in 2-3 Trees», *Acta Informatica*, volumen 14 (1980), páginas 63-86.

[Ellis 1980b] C. S. Ellis, «Concurrent Search and Insertion in AVL Trees», *IEEE Transactions on Computers*, volumen C-29, número 3 (1980), páginas 811-817.

[Ellis 1987] C. S. Ellis, «Concurrency in Linear Hashing», *ACM Transactions on Database Systems*, volumen 12, número 2 (1987), páginas 195-217.

[Elmasri y Larson 1985] R. Elmasri y J. Larson, «A Graphical Query Facility for E-R Databases», *Proc. of the International Conf. on Entity-Relationship Approach* (1985).

[Elmasri y Navathe 2000] R. Elmasri y S. B. Navathe, *Fundamentals of Database Systems*, 3.ª edición, Benjamin Cummings (2000).

[Elmasri andWiederhold 1981] R. Elmasri y G. Wiederhold, «GORDAS: A Formal High-Level Query Language for the Entity-Relationship Model», *Proc. of the International Conf. on Entity-Relationship Approach* (1981).

[Eppinger et al. 1991] J. L. Eppinger, L. B. Mummert y A. Z. Spector, *Camelot and Avalon: A Distributed Transaction Facility*, Morgan Kaufmann (1991).

[Epstein et al. 1978] R. Epstein, M. R. Stonebraker y E. Wong, «Distributed Query Processing in a Relational Database System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1978), páginas 169-180.

[Epstein y Stonebraker 1980] R. Epstein y M. R. Stonebraker, «Analysis of Distributed Database Processing Strategies», *Proc. of the International Conf. on Very Large Databases* (1980), páginas 92-110.

[Escobar-Molano et al. 1993] M. Escobar-Molano, R. Hull y D. Jacobs, «Safety and Translation of Calculus Queries with Scalar Functions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993), páginas 253-264.

[Eswaran et al. 1976] K. P. Eswaran, J. N. Gray, R. A. Lorie e I. L. Traiger, «The Notions of Consistency y Predicate Locks in a Database System», *Communications of the ACM*, volumen 19, número 11 (1976), páginas 624-633.

[Eswaran y Chamberlin 1975] K. P. Eswaran y D. D. Chamberlin, «Functional Specifications of a Subsystem for Database Integrity», *Proc. of the International Conf. on Very Large Databases* (1975), páginas 48-68.

[Evangelidis et al. 1995] G. Evangelidis, D. Lomet y B. Salzberg, «The hB-Pi Tree: A Modified hB-tree Supporting Concurrency, Recovery y Node Consolidation», *Proc. of the International Conf. on Very Large Databases* (1995), páginas 551-561.

[Fagin 1977] R. Fagin, «Multivalued Dependencies and a New Normal Form for Relational Databases», *ACM Transactions on Database Systems,* volumen 2, número 3 (1977), páginas 262-278.

[Fagin 1979] R. Fagin, «Normal Forms and Relational Database Operators», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1979).

[Fagin 1981] R. Fagin, «A Normal Form for Relational Databases That Is Based on Domains and Keys», *ACM Tran-*

sactions on Database Systems, volumen 6, número 3 (1981), páginas 387-415.

[Fagin et al. 1979] R. Fagin, J. Nievergelt, N. Pippenger y H. R. Strong, «Extendible Hashing — A Fast Access Method for Dynamic Files», *ACM Transactions on Database Systems*, volumen 4, número 3 (1979), páginas 315-344.

[Faloutsos y Lin 1995] C. Faloutsos y K.-I. Lin, «Fast Map: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets», en *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 163-174.

[Fayyad et al. 1995] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth y R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, MIT Press (1995).

[Fekete et al. 1990a] A. Fekete, N. Lynch, M. Merritt y W. Weihl, «Commutativity-Based Locking for Nested Transactions», *Journal of Computer and System Science* (1990), páginas 65-156.

[Fekete et al. 1990b] A. Fekete, N. Lynch y W. Weihl, «A Serialization Graph Construction for Nested Transactions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990), páginas 94-108.

[Fernández et al. 1981] E. Fernández, R. Summers y C.Wood, *Database Security and Integrity*, Addison Wesley (1981).

[Fernández et al. 2000] M. F. Fernández, J. Siméon y P. Wadler, «An Algebra for XML Query», *Proc. of the International Conf. on Foundations of Software Technology and Theoretical Computer Science* (2000), páginas 11-45.

[Fernández y Morishima 2001] M. F. Fernández y A. Morishima, «Efficient Evaluation of XML Middle-ware Queries», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2001).

[Finkel y Bentley 1974] R. A. Finkel y J. L. Bentley, «Quad Trees: A Data Structure for Retrieval on Composite Keys», *Acta Informatica,* volumen 4 (1974), páginas 1-9.

[Finkelstein 1982] S. Finkelstein, «Common Expression Analysis in Database Applications», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1982), páginas 235-245.

[Finkelstein et al. 1988] S. Finkelstein, M. Schkolnick y P. Tiberio, «Physical Database Design for Relational Databases», *ACM Transactions on Database Systems,* volumen 13, número 1 (1988), páginas 53-90.

[Fischer 2001] L. Fischer, editor,*Workflow Handbook 2001*, Future Strategies (2001).

[Fischer y Thomas 1983] P. C. Fischer y S. Thomas, «Operators for Non-First-Normal-Form Relations», *Proc. of the International Computer Software Applications Conf.* (1983), páginas 464-475.

[Fishman et al. 1990] D. Fishman, D. Beech, H. Cate, E. Chow, T. Connors, J. Davis, N. Derrett, C. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M. Neimat, T. Ryan y M. Shan. «IRIS: An Object-Oriented Database Management System», en *Zdonik and Maier [1990]*, páginas 216-226.

[Florescu et al. 2000] D. Florescu, D. Kossmann e I. Monalescu, «Integrating keyword search into XML query processing», *Proc. of the International World Wide Web Conf.* (2000), páginas 119-135. También aparece en un número especial de Computer Networks.

[Florescu y Kossmann 1999] D. Florescu y D. Kossmann, «Storing and Querying XML Data Using an RDBMS», *IEEE Data Engineering Bulletin (Special Issue on XML)* (1999), páginas 27-35.

[Franklin et al. 1992] M. J. Franklin, M. J. Zwilling, C. K. Tan, M. J. Carey y D. J. DeWitt, «Crash Recovery in Client-Server EXODUS», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1992), páginas 165-174.

[Franklin et al. 1993] M. J. Franklin, M. Carey y M. Livny, «Local Disk Caching for Client-Server Database Systems», *Proc. of the International Conf. on Very Large Databases* (1993).

[Fredkin 1960] E. Fredkin, «Trie Memory», *Communications of the ACM,* volumen 4, número 2 (1960), páginas 490-499.

[Freedman y DeWitt 1995] C. S. Freedman y D. J. DeWitt, «The SPIFFI Scalable Video-on-Demand Server», *Proc. of theACMSIGMOD Conf. on Management of Data* (1995), páginas 352-363.

[Fry y Sibley 1976] J. Fry y E. Sibley, «Evolution of Data-Base Management Systems», *ACM Computing Survey*, volumen 8, número 1 (1976), páginas 7-42.

[Fushimi et al. 1986] S. Fushimi, M. Kitsuregawa y H. Tanaka, «An Overview of the Systems Software of a Parallel Relational Database Machine: GRACE», *Proc. of the International Conf. on Very Large Databases* (1986).

[Fussell et al. 1981] D. S. Fussell, Z. Kedem y A. Silberschatz, «Deadlock Removal Using Partial Rollback in Database Systems», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1981), páginas 65-73.

[Gadia 1986] S. K. Gadia, «Weak Temporal Relations», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1986), páginas 70-77.

[Gadia 1988] S. K. Gadia, «A Homogeneous Relational Model and Query Language for Temporal Databases», *ACM Transactions on Database Systems,* volumen 13, número 4 (1988), páginas 418-448.

[Galindo-Legaria 1994] C. Galindo-Legaria, «Outerjoins as Disjunctions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1994).

[Galindo-Legaria y Joshi 2001] C. A. Galindo-Legaria y M. M. Joshi, «Orthogonal Optimization of Subqueries and Aggregation», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2001).

[Galindo-Legaria y Rosenthal 1992] C. Galindo-Legaria y A. Rosenthal, «How to Extend a Conventional Optimizer to Handle One-and Two-Sided Outerjoin», *Proc. of the International Conf. on Data Engineering* (1992), páginas 402-409.

[Gallaire y Minker 1978] H. Gallaire y J. Minker, editores, *Logic and Databases*, Plenum Press, Nueva York, NY (1978).

[Gallaire et al. 1984] H. Gallaire, J. Minker y J. M. Nicolas, «Logic and Databases: A Deductive Approach», *ACM Computing Survey,* volumen 16, número 2 (1984).

[Ganguly 1998] S. Ganguly, «Design and Analysis of Parametric Query Optimization Algorithms», *Proc. of the International Conf. on Very Large Databases*, Nueva York, NY(1998).

[Ganguly et al. 1992] S. Ganguly, W. Hasan y R. Krishnamurthy, «Query Optimization for Parallel Execution», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1992).

[Ganguly et al. 1996] S. Ganguly, P. Gibbons, Y. Matias y A. Silberschatz, «A Sampling Algorithm for Estimating

Join Size», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Ganski y Wong 1987] R. A. Ganski y H. K. T. Wong, «Optimization of Nested SQL Queries Revisited», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987).

[García-Molina 1982] H. García-Molina, «Elections in Distributed Computing Systems», *IEEE Transactions on Computers*, volumen C-31, número 1 (1982), páginas 48-59.

[García-Molina 1983] H. García-Molina, «Using Semantic Knowledge for Transaction Processing in a Distributed Database», *ACM Transactions on Database Systems,* volumen 8, número 2 (1983), páginas 186-213.

[García-Molina y Salem 1987] H. García-Molina y K. Salem, «Sagas», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987), páginas 249-259.

[García-Molina y Salem 1992] H. García-Molina y K. Salem, «Main Memory Database Systems: An Overview», *IEEE Transactions on Knowledge and Data Engineering,* volumen 4, número 6 (1992), páginas 509-516.

[García-Molina y Wiederhold 1982] H. García-Molina y G. Wiederhold, «Read-only Transactions in a Distributed Database», *IEEE Transactions on Knowledge and Data Engineering*, volumen 7, número 2 (1982), páginas 209-234.

[Gawlick 1998] D. Gawlick, «Messaging/Queuing in Oracle8», *Proc. of the International Conf. on Data Engineering* (1998), páginas 66-68.

[Geiger 1995] K. Geiger, *Inside ODBC*, Microsoft Press, Redmond, WA (1995).

[Georgakopoulos et al. 1994] D. Georgakopoulos, M. Rusinkiewicz y A. Seth, «Using Tickets to Enforce the Serializability of Multidatabase Transactions», *IEEE Transactions on Knowledge and Data Engineering*, volumen 6, número 1 (1994), páginas 166-180.

[Gifford 1979] D. K. Gifford, «Weighted Voting for Replicated Data», *Proc. the ACM SIGOPS Symposium on Operating Systems Principles* (1979), páginas 150-162.

[Gligor y Shattuck 1980] V. D. Gligor y S. H. Shattuck, «On Deadlock Detection in Distributed Systems», *IEEE Transactions on Software Engineering,* volumen SE-6, número 5 (1980), páginas 435-439.

[Goldberg y Robson 1983] A. Goldberg y D. Robson, *Smalltalk-80: The Language and Its Implementation*, Addison Wesley (1983).

[Goodman 1995] N. Goodman. «An Object-Oriented DBMS War Story: Developing a Genome Mapping Database in C++», en *Kim [1995],* páginas 216-237 (1995).

[Graefe 1990] G. Graefe, «Encapsulation of Parallelism in the Volcano Query Processing System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990), páginas 102-111.

[Graefe 1993] G. Graefe, «Query Evaluation Techniques for Large Databases», *ACM Computing Survey*, volumen 25, número 2 (1993), páginas 73-170.

[Graefe 1995] G. Graefe, «The Cascades Framework for Query Optimization», *Data Engineering Bulletin*, volumen 18, número 3 (1995), páginas 19-29.

[Graefe et al. 1998] G. Graefe, R. Bunker y S. Cooper, «Hash Joins and Hash Teams in Microsoft SQL Server», *Proc. of the International Conf. on Very Large Databases* (1998), páginas 86-97.

[Graefe y McKenna 1993] G. Graefe y W. McKenna, «The Volcano Optimizer Generator», *Proc. of the International Conf. on Data Engineering* (1993), páginas 209-218.

[Graham et al. 1986] M. H. Graham, A. O. Mendelzon y M. Y. Vardi, «Notions of Dependency Satisfaction», *Journal of the ACM,* volumen 33, número 1 (1986), páginas 105-129.

[Gray 1978] J. Gray. «Notes on Data Base Operating System», en *Bayer et al. [1978]*, páginas 393-481 (1978).

[Gray 1981] J. Gray, «The Transaction Concept: Virtues and Limitations», *Proc. of the International Conf. on Very Large Databases* (1981), páginas 144-154.

[Gray 1991] J. Gray, *The Benchmark Handbook for Database and Transaction Processing Systems*, 2.ª edición, Morgan Kaufmann (1991).

[Gray et al. 1975] J. Gray, R. A. Lorie y G. R. Putzolu, «Granularity of Locks and Degrees of Consistency in a Shared Data Base», *Proc. of the International Conf. on Very Large Databases* (1975), páginas 428-451.

[Gray et al. 1976] J. Gray, R. A. Lorie, G. R. Putzolu e I. L. Traiger, *Granularity of Locks and Degrees of Consistency in a Shared Data Base*, Nijssen (1976).

[Gray et al. 1981a] J. Gray, P. Homan, H. F. Korth y R. Obermarck, «A Straw ManAnalysis of the Probability of Waiting and Deadlock», informe técnico, IBM Research Laboratory, San José, Research Report RJ3066 (1981).

[Gray et al. 1981b] J. Gray, P. R. McJones y M. Blasgen, «The Recovery Manager of the System R Database Manager», *ACM Computing Survey,* volumen 13, número 2 (1981), páginas 223-242.

[Gray et al. 1990] J. Gray, B. Horst y M. Walker, «Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 148-161.

[Gray et al. 1995] J. Gray, A. Bosworth, A. Layman y H. Pirahesh, «Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals», informe técnico, Microsoft Research (1995).

[Gray et al. 1996] J. Gray, P. Helland y P. O'Neil, «The Dangers of Replication and a Solution», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 173-182.

[Gray et al. 1997] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow y H. Pirahesh, «Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals», *Data Mining and Knowledge Discovery*, volumen 1, número 1 (1997), páginas 29-53.

[Gray y Edwards 1995] J. Gray y J. Edwards, «Scale Up with TP Monitors», *Byte,* volumen 20, número 4 (1995), páginas 123-130.

[Gray y Graefe 1997] J. Gray y G. Graefe, «The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb», *SIGMODRecord,* volumen 26, número 4 (1997), páginas 63-68.

[Gray y Putzolu 1987] J. Gray y G. R. Putzolu, «The 5 Minute Rule for Trading Memory for Disk Accesses and The 10 Byte Rule for TradingMemory for CPU Time», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987), páginas 395-398.

[Gray y Reuter 1993] J. Gray y A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann (1993).

[Griffin y Libkin 1995] T. Griffin y L. Libkin, «Incremental Maintenance of Views with Duplicates», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995).

[Grossman y Frieder 1998] D. A. Grossman y O. Frieder, *Information Retrieval: Algorithms and Heuristics*, Kluwer Academic (1998).

[Gucht 1987] D. V. Gucht, «On the Expressive Power of the Extended Relational Algebra for the Unnormalized Relational Model», *Proc. of the ACM Symposium on Principles of Database Systems* (1987), páginas 302-312.

[Gupta 1997] H. Gupta, «Selection of Views to Materialize in a Data Warehouse», en *Proc. of the International Conf. on Database Theory* (1997).

[Gupta y Mumick 1995] A. Gupta y I. S. Mumick, «Maintenance of Materialized Views: Problems, Techniques and Applications», *IEEE Data Engineering Bulletin,* volumen 18, número 2 (1995).

[Guttman 1984] A. Guttman, «R-Trees: A Dynamic Index Structure for Spatial Searching», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1984), páginas 47-57.

[H. Lu y Tan 1991] M. S. H. Lu y K. Tan, «Optimization of Multi-Way Join Queries for Parallel Execution», *Proc. of the International Conf. on Very Large Databases* (1991).

[Haas et al. 1989] L. M. Haas, J. C. Freytag, G. M. Lohman y H. Pirahesh, «Extensible Query Processing in Starburst», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1989), páginas 377-388.

[Haas et al. 1990] L. M. Haas, W. Chang, G. M. Lohman, J. McPherson, P. F. Wilms, G. Lapis, B. G. Lindsay, H. Pirahesh, M. J. Carey y E. J. Shekita, «Starburst Mid-Flight: As the dust clears», *IEEE Transactions on Knowledge and Data Engineering*, volumen 2, número 1 (1990), páginas 143-160.

[Haerder y Reuter 1983] T. Haerder y A. Reuter, «Principles of Transaction-Oriented Database Recovery», *ACM Computing Survey,* volumen 15, número 4 (1983), páginas 287-318.

[Haerder y Rothermel 1987] T. Haerder y K. Rothermel, «Concepts for Transaction Recovery in Nested Transactions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987), páginas 239-248.

[Hall 1976] P. A. V. Hall, «Optimization of a Single Relational Expression in a Relational Database System», *IBM Journal of Research and Development,* volumen 20, número 3 (1976), páginas 244-257.

[Halsall 1992] F. Halsall, *Data Communications, Computer Networks, and Open Systems*, Addison Wesley (1992).

[Hammer y McLeod 1975] M. Hammer y D. McLeod, «Semantic Integrity in a Relational Data Base System», *Proc. of the International Conf. on Very Large Databases* (1975), páginas 25-47.

[Hammer y McLeod 1980] M. Hammer y D. McLeod, «Database Description with SDM: A Semantic Data Model», *ACM Transactions on Database Systems,* volumen 6, número 3 (1980), páginas 51-386.

[Hammer y Sarin 1978] M. Hammer y S. Sarin, «Efficient Monitoring of Database Assertions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1978).

[Han y Kamber 2000] J. Han y M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann (2000).

[Harinarayan et al. 1996] V. Harinarayan, J. D. Ullman y A. Rajaraman, «Implementing Data Cubes Efficiently», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Haritsa et al. 1990] J. Haritsa, M. Carey y M. Livny, «On Being Optimistic about Real-Time Constraints», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990).

[Hasan y Motwani 1995] W. Hasan y R. Motwani, «Coloring Away Communication in Parallel Query Optimization», *Proc. of the International Conf. on Very Large Databases* (1995).

[Haskin y Lorie 1982] R. Haskin y R. A. Lorie, «On Extending the Functions of a Relational Database System», *Proc. of the ACM SIGMOD Conf. on Management of Data*, páginas 207-212 (1982).

[Hevner y Yao 1979] A. R. Hevner y S. B. Yao, «Query Processing in Distributed Database Systems», *IEEE Transactions on Software Engineering,* volumen SE-5, número 3 (1979), páginas 177-187.

[Hinrichs 1985] K. H. Hinrichs, *The Grid File System: Implementation and Case Studies of Applications*. Tesis doctoral, Swiss Federal Institute of Technology, Zurich, Switzerland (1985).

[Hollinsworth 1994] D. Hollinsworth, *The Work-flow Reference Model*, Workflow Management Coalition, TC00-1003 (1994).

[Holt 1971] R. C. Holt, «Comments on Prevention of System Deadlocks», *Communications of the ACM*, volumen 14, número 1 (1971), páginas 36-38.

[Holt 1972] R. C. Holt, «Some Deadlock Properties of Computer Systems», *ACM Computing Survey*, volumen 4, número 3 (1972), páginas 179-196.

[Hong et al. 1993] D. Hong, T. Johnson y S. Chakravarthy, «Real-Time Transaction Scheduling: A Cost Conscious Approach», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993).

[Hong y Stonebraker 1991] W. Hong y M. Stonebraker, «Optimization of Parallel Query Execution Plans in XPRS», *Proc. of the International Symposium on Parallel and Distributed Information Systems* (1991), páginas 218-225.

[Howes et al. 1999] T. A. Howes, M. C. Smith y G. S. Good, *Understanding and Deploying LDAP Directory Services*, Macmillan Publishing, Nueva York (1999).

[Hsu e Imielinski 1985] A. Hsu y T. Imielinski, «Integrity Checking for Multiple Updates», en *Proc. of the ACM SIGMOD Conf. on Management of Data* (1985), páginas 152-168.

[Huang y García-Molina 2001] Y. Huang y H. García-Molina, «Exactly-once Semantics in a Replicated Messaging System», *Proc. of the International Conf. on Data Engineering* (2001), páginas 3-12.

[Hudson y King 1989] S. E. Hudson y R. King, «Cactis:A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System», *ACM Transactions on Database Systems*, volumen 14, número 3 (1989), páginas 291-321.

[Huffman 1993] A. Huffman, «Transaction Processing with TUXEDO», *Proc. of the International Symposium on Parallel and Distributed Information Systems* (1993).

[IBM 1978] *Query-by Example Terminal Users Guide*. IBM Corporation. IBM Form Número SH20-20780 (1978).

[IBM 1987] IBM, «Systems Application Architecture: Common Programming Interface, Database Reference», informe técnico, IBM Corporation, IBM Form Número SC26-4348-0 (1987).

[Imielinski et al. 1995] T. Imielinski, S. Viswanathan y B. R. Badrinath, «Energy Efficient Indexing on the Air», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 25-36.

[Imielinski y Badrinath 1994] T. Imielinski y B. R. Badrinath, «Mobile Computing — Solutions and Challenges», *Communications of the ACM,* volumen 37, número 10 (1994).

[Imielinski y Korth 1996] T. Imielinski y H. F. Korth, editores, *Mobile Computing*, Kluwer Academic Publishers (1996).

[Ioannidis et al. 1992] Y. E. Ioannidis, R. T. Ng, K. Shim y T. K. Sellis, «Parametric Query Optimization», *Proc. of the International Conf. on Very Large Databases* (1992), páginas 103-114.

[Ioannidis y Christodoulakis 1993] Y. Ioannidis y S. Christodoulakis, «Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results», *ACM Transactions on Database Systems*, volumen 18, número 4 (1993), páginas 709-748.

[Ioannidis y Kang 1990] Y. Ioannidis y Y. Kang. «Randomized Algorithms for Optimizing Large Join Queries», en *Proc. of the ACM SIGMOD Conf. on Management of Data*, páginas 312-321 (1990).

[Ioannidis y Poosala 1995] Y. E. Ioannidis y V. Poosala. «Balancing Histogram Optimality and Practicality for Query Result Size Estimation», *Proc. of the ACM SIGMOD Conf. on Management of Data*, páginas 233-244 (1995).

[Ioannidis y Wong 1987] Y. E. Ioannidis y E. Wong. «Query Optimization by Simulated Annealing», *Proc. of the ACM SIGMOD Conf. on Management of Data*, páginas 9-22 (1987).

[Ishikawa et al. 1993] H. Ishikawa, F. Suzuki, F. Kozakura, A. Makinouchi, M. Miyagishima, Y. Izumida, M. Aoshima y Y. Yamana, «The Model, Language, and Implementation of an Object-Oriented Multimedia Knowledge Base Management System», *ACM Transactions on Database Systems*, volumen 18, número 1 (1993), páginas 1-50.

[Jaeschke y Schek 1982] G. Jaeschke y H. J. Schek. «Remarks on the Algebra of Non First Normal Form Relations», *Proc. of the ACM SIGMOD Conf. on Management of Data*, páginas 124-138 (1982).

[Jagadish et al. 1993] H. V. Jagadish, A. Silberschatz y S. Sudarshan, «Recovering from Main-Memory Lapses», *Proc. of the International Conf. on Very Large Databases* (1993).

[Jagadish et al. 1994] H. Jagadish, D. Lieuwen, R. Rastogi, A. Silberschat y S. Sudarshan, *Dali: A High Performance Main Memory Storage Manager* (1994).

[Jain y Dubes 1988] A. K. Jain y R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall (1988).

[Jajodia y Sandhu 1990] S. Jajodia y R. Sandhu, «Polyinstantiation Integrity in Multilevel Relations», *Proc. of the IEEE Symposium on Research in Security and Privacy* (1990), páginas 104-115.

[Jarke y Koch 1984] M. Jarke y J. Koch, «Query Optimization in Database Systems», *ACM Computing Survey*, volumen 16, número 2 (1984), páginas 111-152.

[Jensen et al. 1994] C. S. Jensen et al., «A Consensus Glossary of Temporal Database Concepts», *ACM Sigmod Record,* volumen 23, número 1 (1994), páginas 52-64.

[Jhingran et al. 1997] A. Jhingran, T. Malkemus y S. Padmanabhan, «Query Optimization in DB2 Parallel Edition», *Data Engineering Bulletin,* volumen 20, número 2 (1997), páginas 27-34.

[Jin et al. 1993] W. Jin, L. Ness, M. Rusinkiewicz y A. Sheth. «Concurrency Control and Recovery of Multidatabase-Work Flows in Telecommunication Applications», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993).

[Johnson 1999a] T. Johnson, «Performance Measurements of Compressed Bitmap Indices», en *Proc. of the International Conf. on Very Large Databases* (1999).

[Johnson 1999b] T. Johnson, «Performance Measurements of Compressed Bitmap Indices», en *Proc. of the International Conf. on Very Large Databases* (1999).

[Johnson y Shasha 1993] T. Johnson y D. Shasha, «The Performance of Concurrent B-Tree Algorithms», *ACM Transactions on Database Systems,* volumen 18, número 1 (1993).

[Jones y Willet 1997] K. S. Jones y P. Willet, editores, *Readings in Information Retrieval*, Morgan Kaufmann (1997).

[Joshi 1991] A. Joshi, «Adaptive Locking Strategies in a Multi-Node Shared Data Model Environment», *Proc. of the International Conf. on Very Large Databases* (1991).

[Joshi et al. 1998] A. Joshi,W. Bridge, J. Loaiza y T. Lahiri, «Checkpointing in Oracle», *Proc. of the International Conf. on Very Large Databases* (1998), páginas 665-668.

[Kaiser 1990] G. Kaiser, «A Flexible Transaction Model for Software Engineering», en *IEEE Transactions on Knowledge and Data Engineering* (1990).

[Kambayashi et al. 1982] Y. Kambayashi, M. Yoshikawa y S. Yajima, «Query Processing for Distributed Databases Using Generalized Semijoins», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1982), páginas 151-160.

[Kamel y Faloutsos 1992] I. Kamel y C. Faloutsos. «Parallel -Trees», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1992).

[Kanne andMoerkotte 2000] C.-C. Kanne y G. Moerkotte, «Efficient Storage of XML Data», *Proc. of the International Conf. on Data Engineering* (2000), página 198.

[Kapitskaia et al. 2000] O. Kapitskaia, R. T. Ng y D. Srivastava, «Evolution and Revolutions in LDAP Directory Caches», *Proc. of the International Conf. on Extending Database Technology* (2000), páginas 202-216.

[Kedem y Silberschatz 1979] Z. M. Kedem y A. Silberschatz, «Controlling Concurrency Using Locking Protocols», *Proc. of the Annual IEEE Symposium on Foundations of Computer Science* (1979), páginas 275-285.

[Kedem y Silberschatz 1983] Z. M. Kedem y A. Silberschatz, «Locking Protocols: From Exclusive to Shared Locks», *ACM Press,* volumen 30, número 4 (1983), páginas 787-804.

[Khoshafian y Copeland 1990] S. Khoshafian y G. P. Copeland. «Object Identity», en *Zdonik and Maier [1990],* páginas 37-46 (1990).

[Kifer et al. 1992] M. Kifer, W. Kim y Y. Sagiv, «Querying Object Oriented Databases», en *Proc. of the ACM SIGMOD Conf. onManagement of Data* (1992), páginas 393-402.

[Kim 1982] W. Kim, «On Optimizing an SQL-like Nested Query», *ACM Transactions on Database Systems*, volumen 3, número 3 (1982), páginas 443-469.

[Kim 1984] W. Kim. «Query Optimization for Relational Database Systems», en *Unger et al. [1984]* (1984).

[Kim 1990] W. Kim, *Introduction to Object-Oriented Databases*, MIT Press, Cambridge (1990).

[Kim 1995] W. Kim, editor, *Modern Database Systems*, ACM Press/ Addison Wesley (1995).

[Kim et al. 1985] W. Kim, D. S. Reiner y D. S. Batory, editores, *Query Processing in Database Systems*, Springer Verlag (1985).

[Kim et al. 1988] W. Kim, N. Ballou, J. Banerjee, H. T. Chou, J. F. Garza y D. Woelk, «Integrating an Object-Oriented Programming System with a Database System», *Proc. of the International Conf. on Object-Oriented Programming Systems, Languages, and Applications* (1988).

[Kim y Lochovsky 1989] W. Kim y F. Lochovsky, editores, *Object-Oriented Concepts, Databases, and Applications*, Addison Wesley (1989).

[King 1981] J. J. King, «QUIST: A System for Semantic Query Optimization in Relational Data Bases», *Proc. of the International Conf. on Very Large Databases* (1981), páginas 510-517.

[King et al. 1991] R. P. King, N. Halim, H. García-Molina y C. Polyzois, «Management of a Remote Backup Copy for Disaster Recovery», *ACM Transactions on Database Systems*, volumen 16, número 2 (1991), páginas 338-368.

[Kirchmer 1999] M. Kirchmer, *Business Process Oriented Implementation of Standard Software*: *How to Achieve Competitive Advantage Efficiently and Effectively*, 2.ª edición, Springer Verlag (1999).

[Kitsuregawa et al. 1983] M. Kitsuregawa, H. Tanaka y T. MotoOka, «Application of Hash to a Database Machine and its Architecture», *New Generation Computing, número* 1 (1983), páginas 62-74.

[Kitsuregawa y Ogawa 1990] M. Kitsuregawa e Y. Ogawa, «Bucket Spreading Parallel Hash: A New, Robust, Parallel Hash Join Method for Skew in the Super Database Computer», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 210-221.

[Kleinberg 1999] J. M. Kleinberg, «Authoritative Sources in a Hyperlinked Environment», *Journal of the ACM*, volumen 46, número 5 (1999), páginas 604-632.

[Kleinrock 1975] L. Kleinrock, *Queuing Systems, volumen 1: Theory*, John Wiley and Sons (1975).

[Kleinrock 1976] L. Kleinrock, *Queuing Systems, volumen 2: Computer Applications*, John Wiley and Sons (1976).

[Klug 1982] A. Klug, «Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions», *ACM Press,* volumen 29, número 3 (1982), páginas 699-717.

[Knapp 1987] E. Knapp, «Deadlock Detection in Distributed Databases», *ACM Computing Survey*, volumen 19, número 4 (1987).

[Knuth 1973] D. E. Knuth, *The Art of Computer Programming*, volumen 3, Addison Wesley, Sorting and Searching (1973).

[Kohavi y Provost 2001] R. Kohavi y F. Provost, editores, *Applications of Data Mining to Electronic Commerce*, Kluwer Academic Publishers (2001).

[Kohler 1981] W. H. Kohler, «A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems», *ACM Computing Survey,* volumen 13, número 2 (1981), páginas 149-183.

[Konstan et al. 1997] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon y J. Riedl, «GroupLens: Applying Collaborative Filtering to Usenet News», *Communications of the ACM,* volumen 40, número 3 (1997), páginas 77-87.

[Korth 1982] H. F. Korth, «Deadlock Freedom Using Edge Locks», *ACM Transactions on Database Systems*, volumen 7, número 4 (1982), páginas 632-652.

[Korth 1983] H. F. Korth, «Locking Primitives in a Database System», *Journal of the ACM*, volumen 30, número 1 (1983), páginas 55-79.

[Korth et al. 1988] H. F. Korth, W. Kim y F. Bancilhon, «On Long Duration CAD Transactions», *Information Science,* volumen 46 (1988), páginas 73-107.

[Korth et al. 1990a] H. F. Korth, E. Levy y A. Silberschatz, «A Formal Approach to Recovery by Compensating Transactions», *Proc. of the International Conf. on Very Large Databases* (1990).

[Korth et al. 1990b] H. F. Korth, N. Soparkar y A. Silberschatz, «Triggered Real-Time Databases with Consistency Constraints», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 71-82.

[Korth y Speegle 1988] H. F. Korth y G. Speegle. «Formal Model of Correctness Without Serializability», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1988).

[Korth y Speegle 1990] H. F. Korth y G. Speegle, «Long Duration Transactions in Software Design Projects», *Proc. of the International Conf. on Data Engineering* (1990), páginas 568-575.

[Korth y Speegle 1994] H. F. Korth y G. Speegle, «Formal Aspects of Concurrency Control in Long Duration Transaction Systems Using the NT/PV Model», *ACM Transactions on Database Systems*, volumen 19, número 3 (1994), páginas 492-535.

[Kumar y Stonebraker 1988] A. Kumar y M. Stonebraker. «Semantics Based Transaction Management Techniques for Replicated Data», *Proc. of the ACM SIGMOD Conf. on Management of Data*, páginas 117-125 (1988).

[Kung y Lehman 1980] H. T. Kung y P. L. Lehman, «Concurrent Manipulation of Binary Search Trees», *ACM Transactions on Database Systems,* volumen 5, número 3 (1980), páginas 339-353.

[Kung y Robinson 1981] H. T. Kung y J. T. Robinson, «Optimistic Concurrency Control», *ACM Transactions on Database Systems*, volumen 6, número 2 (1981), páginas 312-326.

[Labio et al. 1997] W. Labio, D. Quass y B. Adelberg, «Physical Database Design for Data Warehouses», *Proc. of the International Conf. on Data Engineering* (1997).

[Lahiri et al. 2001] T. Lahiri, A. Ganesh, R. Weiss y A. Joshi, «Fast-Start: Quick Fault Recovery in Oracle», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2001).

[Lai y Wilkinson 1984] M. Y. Lai y W. K. Wilkinson, «Distributed Transaction Management in JASMIN», *Proc. of the International Conf. on Very Large Databases* (1984), páginas 466-472.

[Lamb et al. 1991] C. Lamb, G. Landis, J. Orenstein y D. Weinreb, «The ObjectStore Database System», *Communications of the ACM,* volumen 34, número 10 (1991), páginas 51-63.

[Lamport 1978] L. Lamport, «Time, Clocks, and the Ordering of Events in a Distributed System», *Communications of the ACM,* volumen 21, número 7 (1978), páginas 558-565.

[Lampson y Sturgis 1976] B. Lampson y H. Sturgis, «Crash Recovery in a Distributed Data Storage System», infor-

me técnico, Computer Science Laboratory, Xerox Palo Alto Research Center, Palo Alto (1976).

[Langerak 1990] R. Langerak, «View Updates in Relational Databases with an Independent Scheme», *ACM Transactions on Database Systems,* volumen 15, número 1 (1990), páginas 40-66.

[Lanzelotte et al. 1993] R. Lanzelotte, P. Valduriez y M. Zar, «On the Effectiveness of Optimization Search Strategies for Parallel Execution Spaces», *Proc. of the International Conf. on Very Large Databases* (1993).

[Larson 1978] P. Larson, «Dynamic Hashing», *BIT,* volumen 18 (1978).

[Larson 1982] P. Larson, «Performance Analysis of Linear Hashing with Partial Expansions», *ACM Transactions on Database Systems,* volumen 7, número 4 (1982), páginas 566-587.

[Larson 1988] P. Larson, «Linear Hashingwith Separators — A Dynamic Hashing Scheme Achieving One-Access Retrieval», *ACM Transactions on Database Systems,* volumen 19, número 3 (1988), páginas 366-388.

[Larson y Yang 1985] P. Larson y H. Z. Yang, «Computing Queries from Derived Relations», en *Proc. of the International Conf. on Very Large Databases* (1985), páginas 59-269.

[Lecluse et al. 1988] C. Lecluse, P. Richard y F. Velez, «O2: An Object-Oriented Data Model», *Proc. of the International Conf. on Very Large Databases* (1988), páginas 424-433.

[Lee y Liou 1996] S. Y. Lee y R. L. Liou, «A Multi-Granularity Locking Model for Concurrency Control in Object-Oriented Database Systems», *IEEE Transactions on Knowledge and Data Engineering*, volumen 8, número 1 (1996), páginas 144-156.

[Lehman y Yao 1981] P. L. Lehman y S. B. Yao, «Efficient Locking for Concurrent Operations on B-trees», *ACM Transactions on Database Systems,* volumen 6, número 4 (1981), páginas 650-670.

[Lenzerini y Santucci 1983] M. Lenzerini y C. Santucci. «Cardinality Constraints in the Entity Relationship Model», en *Davis et al. [1983]* (1983).

[Lien y Weinberger 1978] Y. E. Lien y P. J. Weinberger. «Consistency, Concurrency and Crash Recovery», *Proc. of the ACM SIGMOD Conf. on Management of Data*, páginas 9-14 (1978).

[Lin et al. 1994] E. T. Lin, E. R. Omiecinski y S. Yalamanchili, «Large Join Optimization on a Hypercube Multiprocessor», *IEEE Transactions on Knowledge and Data Engineering*, volumen 6, número 2 (1994), páginas 304-315.

[Lindsay et al. 1980] B. G. Lindsay, P. G. Selinger, C. Galtieri, J. N. Gray, R. A. Lorie, T. G. Price, G. R. Putzolu, I. L. Traiger y B. W. Wade. «Notes on Distributed Databases», en Draffen y P. 431, editores, *Distributed Data Bases*, páginas 247-284. Cambridge University Press, Cambridge, Inglaterra (1980).

[Litwin 1978] W. Litwin, «Virtual Hashing: A Dynamically Changing Hashing», *Proc. of the International Conf. on Very Large Databases* (1978), páginas 517-523.

[Litwin 1980] W. Litwin, «Linear Hashing: A New Tool for File and Table Addressing», *Proc. of the International Conf. on Very Large Databases* (1980), páginas 212-223.

[Litwin 1981] W. Litwin. «Trie Hashing», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1981), páginas 19-29.

[Liu et al. 2000] L. Liu, C. Pu y W. Han, «XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources», *Proc. of the International Conf. on Data Engineering* (2000), páginas 611-621.

[Lo y Ravishankar 1996] M.-L. Lo y C. V. Ravishankar. «Spatial Hash-Joins», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Loeb 1998] L. Loeb, *Secure Electronic Transactions*: *Introduction and Technical Reference*, ArtechHouse (1998).

[Lomet 1981] D. G. Lomet, «Digital B-trees», *Proc. of the International Conf. on Very Large Databases* (1981), páginas 333-344.

[Lomet y Salzberg 1992] D. Lomet y B. Salzberg. «Access Method Concurrency with Recovery», *Proc. of the ACM SIGMOD Conf. on Management of Data*, páginas 351-360. También aparece en The VLDB Journal (1997) (1992).

[Lopresti y Tomkins 1993] D. P. Lopresti y A. Tomkins, «Approximate Matching of Hand Drawn Pictograms», *Proc. of the INTERCHI 93 Conf.* (1993).

[Lorie 1977] R. A. Lorie, «Physical Integrity in a Large Segmented Database», *ACM Transactions on Database Systems*, volumen 2, número 1 (1977), páginas 91-104.

[Lorie et al. 1985] R. Lorie, W. Kim, D. McNabb, W. Plouffe y A. Meier. «Supporting Complex Objects in a Relational System for Engineering Databases», en *Kim et al. [1985],* páginas 145-155 (1985).

[Lunt 1995] T. F. Lunt. «Authorization in Object-Oriented Databases», en *Kim [1995]*, (1995), páginas 130-145.

[Lynch 1983] N. A. Lynch, «Multilevel Atomicity-A New Correctness Criterion for Database Concurrency Control», *ACM Transactions on Database Systems,* volumen 8, número 4 (1983), páginas 484-502.

[Lynch et al. 1988] N. A. Lynch, M. Merritt, W. Weihl y A. Fekete, «A Theory of Atomic Transactions», *Proc. of the International Conf. on Database Theory* (1988), páginas 41-71.

[Lynch y Merritt 1986] N. A. Lynch y M. Merritt, «Introduction to the Theory of Nested Transactions», *Proc. of the International Conf. on Database Theory* (1986).

[Lyngbaek y Vianu 1987] P. Lyngbaek y V. Vianu. «Mapping a Semantic Database Model to the Relational Model», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987), páginas 132-142.

[Mackert y Lohman 1986] L. F. Mackert y G. M. Lohman, «R* Optimizer Validation and Performance Evaluation for Distributed Queries», *Proc. of the International Conf. on Very Large Databases* (1986).

[Maier 1983] D. Maier, *The Theory of Relational Databases*, Computer Science Press, Rockville (1983).

[Maier et al. 1986] D. Maier, J. Stein, A. Otis y A. Purdy, «Development of an Object-Oriented DBMS», *Proc. of the Object-Oriented Programming Languages, Systems and Applications Conf. (OOPSLA)* (1986), páginas 472-482.

[Maier y Stein 1986] D. Maier y J. Stein, «Indexing in an Object-Oriented DBMS», *Proc. of the International Workshop on Object-Oriented Database Systems* (1986).

[Makinouchi 1977] A. Makinouchi, «A Consideration of Normal Form on Not-necessarily Normalized Relations in the Relational Data Model», *Proc. of the International Conf. on Very Large Databases* (1977), páginas 447-453.

[Markowitz y Raz 1983] V. Markowitz e Y. Raz. «ERROL: An Entity-Relationship, Role Oriented, Query Language», en *Davis et al. [1983]* (1983).

[Markowitz y Shoshani 1992] V. M. Markowitz y A. Shoshani, «Represented Extended Entity-Relationship Structures in Relational Databases», *ACM Transactions on Database Systems*, volumen 17 (1992), páginas 385-422.

[Martin et al. 1989] J. Martin, K. K. Chapman y J. Leben, *DB2: Concepts, Design, and Programming*, Prentice Hall (1989).

[Mattison 1996] R. Mattison, *Data Warehousing: Strategies, Technologies, and Techniques*, McGraw Hill (1996).

[McCarthy y Dayal 1989] D. McCarthy y U. Dayal, «The Architecture of an Active Database Management System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1989), páginas 215-224.

[McCune y Henschen 1989] W. W. McCune y L. J. Henschen, «Maintaining State Constraints in Relational Databases:A Proof Theoretic Basis», *ACM Transactions on Database Systems*, volumen 36, número 1 (1989), páginas 46-68.

[McHugh y Widom 1999] J. McHugh y J. Widom, «Query Optimization for XML», *Proc. of the International Conf. on Very Large Databases* (1999), páginas 315-326.

[Mehrotra et al. 1991] S. Mehrotra, R. Rastogi, H. F. Korth y A. Silberschatz, «Non-Serializable Executions in Heterogeneous Distributed Database Systems», *Proc. of the First International Conf. on Parallel and Distributed Information Systems* (1991).

[Mehrotra et al. 1992a] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth y A. Silberschatz. «Ensuring Transaction Atomicity in Multidatabase Systems», en *Proc. of the ACM Symposium on Principles of Database Systems* (1992).

[Mehrotra et al. 1992b] S. Mehrotra, R. Rastogi, H. F. Korth, A. Silberschatz e Y. Breitbart. «The Concurrency Control Problem in Multidatabases: Characteristics and Solutions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1992).

[Melton y Eisenberg 2000] J. Melton y A. Eisenberg, *Understanding SQL and Java Together: A Guide to SQLJ, JDBC, and Related Technologies*, Morgan Kaufmann (2000).

[Melton y Simon 1993] J. Melton y A. R. Simon, *Understanding The New SQL: A Complete Guide*, Morgan Kaufmann (1993).

[Menasce et al. 1980] D. A. Menasce, G. Popek y R. Muntz, «A Locking Protocol for Resource Coordination in Distributed Databases», *ACM Transactions on Database Systems*, volumen 5, número 2 (1980), páginas 103-138.

[Menasce y Muntz 1979] D. A. Menasce y R. R. Muntz, «Locking and Deadlock Detection in Distributed Databases», *IEEE Transactions on Software Engineering,* volumen SE-5, número 3 (1979), páginas 195-202.

[Microsoft 1997] Microsoft, *Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference*, Microsoft Press (1997).

[Mistry et al. 2001] H. Mistry, P. Roy, S. Sudarshan y K. Ramamritham, «Materialized View Selection and Maintenance Using Multi-Query Optimization», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2001).

[Mitchell 1997] T. M. Mitchell, *Machine Learning*, McGraw Hill (1997).

[Mohan 1990a] C. Mohan, «ARIES/KVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operations on B-Tree indexes», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 392-405.

[Mohan 1990b] C. Mohan, «Commit-LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 406-418.

[Mohan 1993] C. Mohan. «IBM's Relational Database Products: Features and Technologies», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993).

[Mohan et al. 1986] C. Mohan, B. Lindsay y R. Obermarck, «Transaction Management in the R* Distributed Database Management System», *ACM Transactions on Database Systems*, volumen 11, número 4 (1986), páginas 378-396.

[Mohan et al. 1992] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh y P. Schwarz, «ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging», *ACM Transactions on Database Systems*, volumen 17, número 1 (1992).

[Mohan y Levine 1992] C. Mohan y F. Levine, «ARIES/IM: An Efficient and High-Concurrency Index Management Method Using Write-Ahead Logging», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1992).

[Mohan y Lindsay 1983] C. Mohan y B. Lindsay, «Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions», *Proc. of the ACM Symposium on Principles of Distributed Computing* (1983).

[Mohan y Narang 1991] C. Mohan e I. Narang, «Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment», *Proc. of the International Conf. on Very Large Databases* (1991).

[Mohan y Narang 1992] C. Mohan e I. Narang, «Efficient Locking and Caching of Data in the Multisystem Shared Disks Transaction Environment», *Proc. of the International Conf. on Extending Database Technology* (1992).

[Mohan y Narang 1994] C. Mohan e I. Narang. «ARIES/CSA: A Method for Database Recovery in Client-Server Architectures», *Proc. of the ACM SIGMOD Conf. on Management of Data,* páginas 55-66 (1994).

[Mok et al. 1996] W. Y. Mok, Y.-K. Ng y D. W. Embley, «A Normal Form for Precisely Characterizing Redundancy in Nested Relations», *ACM Transactions on Database Systems*, volumen 21, número 1 (1996), páginas 77-106.

[Moss 1982] J. E. B. Moss, «Nested Transactions and Reliable Distributed Computing», *Proc. of the Symposium on Reliability in Distributed Software and Database Systems* (1982).

[Moss 1985] J. E. B. Moss, *Nested Transactions: An Approach to Reliable Distributed Computing*, MIT Press, Cambridge (1985).

[Moss 1987] J. E. B. Moss, «Log-Based Recovery for Nested Transactions», *Proc. of the International Conf. on Very Large Databases* (1987), páginas 427-432.

[Moss 1990] J. E. B. Moss, «Working with Objects: To Swizzle or Not to Swizzle», informe técnico, Computer and Information Science, University of Massachusetts, Amherst, informe técnico COINS TR 90-38 (1990).

[Mumick et al. 1996] I. S. Mumick, S. Finkelstein, H. Pirahesh y R. Ramakrishnan, «Magic Conditions», *ACM Transactions on Database Systems,* volumen 21, número 1 (1996), páginas 107-155.

[Nakayama et al. 1984] T. Nakayama, M. Hirakawa y T. Ichikawa, «Architecture and Algorithm for Parallel Execution of a Join Operation», *Proc. of the International Conf. on Data Engineering* (1984).

[Naqvi y Tsur 1988] S. Naqvi y S. Tsur, *A Logic Language for Data and Knowledge Bases*, Computer Science Press, Rockville (1988).

[Ng y Han 1994] R. T. Ng y J. Han, «Efficient and Effective Clustering Methods for Spatial Data Mining», *Proc. of the International Conf. on Very Large Databases* (1994).

[Nievergelt et al. 1984] J. Nievergelt, H. Hinterberger y K. C. Sevcik, «The Grid File: An Adaptable Symmetric Multikey File Structure», *ACM Transactions on Database Systems*, volumen 9, número 1 (1984), páginas 38-71.

[North 1995] K. North, *Windows Multi-DBMS Programming: Using C++, Visual Basic, ODBC, OLE2, and Tools for DBMS Projects*, John Wiley and Sons (1995).

[Obermarck 1982] R. Obermarck, «Distributed Deadlock Detection Algorithm», *ACM Transactions on Database Systems*, volumen 7, número 2 (1982), páginas 187-208.

[O'Neil y O'Neil 2000] P. O'Neil y E. O'Neil, *Database: Principles, Programming, Performance*, 2.ª edición, Morgan Kaufmann (2000).

[O'Neil y Quass 1997] P. O'Neil y D. Quass, «Improved Query Performance with Variant Indexes», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1997).

[Oracle 1997] *Oracle 8 Concepts Manual*. Oracle Corporation, Redwood Shores (1997).

[Orenstein 1982] J. A. Orenstein, «Multidimensional Tries Used for Associative Searching», *Information Processing Letters,* volumen 14, número 4 (1982), páginas 150-157.

[Ozcan et al. 1997] F. Ozcan, S. Nural, P. Koksal, C. Evrendilek y A. Dogac, «Dynamic Query Optimization in Multidatabases», *Data Engineering Bulletin,* volumen 20, número 3 (1997), páginas 38-45.

[Ozden et al. 1994] B. Ozden, A. Biliris, R. Rastogi y A. Silberschatz, «A Low-cost Storage Server for a Movie on Demand Database», *Proc. of the International Conf. on Very Large Databases* (1994).

[Ozden et al. 1995a] B. Ozden, R. Rastogi y A. Silberschatz, «A Framework for the Storage and Retrieval of Continuous Media Data», *Proc. of the IEEE International Conf. on Multimedia Computing and Systems* (1995).

[Ozden et al. 1995b] B. Ozden, R. Rastogi y A. Silberschatz, «Research Issues in Multimedia Storage Servers», *ACM Computing Survey,* volumen 27, número 4 (1995), páginas 617-620.

[Ozden et al. 1996a] B. Ozden, R. Rastogi, P. Shenoy y A. Silberschatz. «Fault-Tolerant Architectures for Continuous Media Servers», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Ozden et al. 1996b] B. Ozden, R. Rastogi y A. Silberschatz, «On the Design of a Low-Cost Video-on-Demand Storage System», *Multimedia Systems Journal,* volumen 4, número 1 (1996), páginas 40-54.

[Ozsoyoglu et al. 1987] G. Ozsoyoglu, Z. M. Ozsoyoglu y V.Matos, «Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions», *ACM Transactions on Database Systems,* volumen 12, número 4 (1987), páginas 566-592.

[Ozsoyoglu y Snodgrass 1995] G. Ozsoyoglu y R. Snodgrass, «Temporal and Real-Time Databases: A Survey», *IEEE Transactions on Knowledge and Data Engineering,* volumen 7, número 4 (1995), páginas 513-532.

[Ozsoyoglu y Yuan 1987] G. Ozsoyoglu y L. Yuan, «Reduced MVDs and Minimal Covers», *ACM Transactions on Database Systems*, volumen 12, número 3 (1987), páginas 377-394.

[Ozsu y Valduriez 1999] T. Ozsu y P. Valduriez, *Principles of Distributed Database Systems*, 2.ª edición, Prentice Hall (1999).

[Pang et al. 1995] H.-H. Pang, M. J. Carey y M. Livny, «Multiclass Scheduling in Real-Time Database Systems», *IEEE Transactions on Knowledge and Data Engineering*, volumen 2, número 4 (1995), páginas 533-551.

[Papadimitriou 1979] C. H. Papadimitriou, «The Serializability of Concurrent Database Updates», *Journal of the ACM,* volumen 26, número 4 (1979), páginas 631-653.

[Papadimitriou 1982] C. H. Papadimitriou, «A Theorem in Database Concurrency Control», *Journal of the ACM*, volumen 29, número 5 (1982), páginas 998-1006.

[Papadimitriou 1986] C. H. Papadimitriou, *The Theory of Database Concurrency Control*, Computer Science Press, Rockville (1986).

[Papadimitriou et al. 1977] C. H. Papadimitriou, P. A. Bernstein y J. B. Rothnie, «Some Computational Problems related to Database Concurrency Control», *Proc. of the Conf. on Theoretical Computer Science* (1977), páginas 275-282.

[Papakonstantinou et al. 1996] Y. Papakonstantinou, A. Gupta y L. Haas, «Capabilities-Based Query Rewriting in Mediator Systems», *Proc. of the International Conf. on Parallel and Distributed Information Systems* (1996).

[Parker et al. 1983] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser y C. Kline, «Detection of Mutual Inconsistency in Distributed Systems», *IEEE Transactions on Software Engineering*, volumen 9, número 3 (1983), páginas 240-246.

[Patel y DeWitt 1996] J. Patel y D. J. DeWitt, «Partition Based Spatial-Merge Join», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Patterson et al. 1988] D. A. Patterson, G. Gibson y R. H. Katz, «A Case for Redundant Arrays of Inexpensive Disks (RAID)», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1988), páginas 109-116.

[Patterson y Hennessy 1995] D. A. Patterson y J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, 2.ª edición, Morgan Kaufmann (1995).

[Pellenkoft et al. 1997] A. Pellenkoft, C. A. Galindo-Legaria y M. Kersten, «The Complexity of Transformation-Based Join Enumeration», *Proc. of the International Conf. on Very Large Databases*, Atenas, Grecia (1997), páginas 306-315.

[Penney y Stein 1987] J. Penney y J. Stein, «Class Modification in the GemStone Object-Oriented DBMS», *Proc. of the Object-Oriented Programming Languages, Systems and Applications Conf. (OOPSLA)* (1987).

[Pless 1989] V. Pless, *Introduction to the Theory of Error-Correcting Codes*, 2.ª edición, John Wiley and Sons (1989).

[Poe 1995] V. Poe, *Building a Data Warehouse for Decision Support*, Prentice Hall (1995).

[Poess y Floyd 2000] M. Poess y C. Floyd, «New TPC Benchmarks for Decision Support and Web Commerce», *ACM SIGMOD Record,* volumen 29, número 4 (2000).

[Polyzois y García-Molina 1994] C. Polyzois y H. García-Molina, «Evaluation of Remote Backup Algorithms for Transaction-Processing Systems», *ACM Transactions on Database Systems*, volumen 19, número 3 (1994), páginas 423-449.

[Poosala et al. 1996] V. Poosala, Y. E. Ioannidis, P. J. Haas y E. J. Shekita, «Improved Histograms for Selectivity Estimation of Range Predicates», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 294-305.

[Popek et al. 1981] G. J. Popek, B. J. Walker, J. M. Chow, D. Edwards, C. Kline, G. Rudisin y G. Thiel, «LOCUS: A Network Transparent, High Reliability Distributed System», *Proc. of the Eighth Symposium on Operating System Principles* (1981), páginas 169-177.

[Pu et al. 1988] C. Pu, G. Kaiser y N. Hutchinson, «Split-transactions for Open-Ended Activities», *Proc. of the International Conf. on Very Large Databases* (1988), páginas 26-37.

[Qian y Lunt 1996] X. Qian y T. F. Lunt, «A MAC Policy Framework for Multilevel Relational Databases», *IEEE Transactions on Knowledge and Data Engineering*, volumen 8, número 1 (1996), páginas 3-15.

[Rahm 1993] E. Rahm, «Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems», *ACM Transactions on Database Systems*, volumen 8, número 2 (1993).

[Ramakrishna y Larson 1989] M. V. Ramakrishna y P. Larson, «File Organization Using Composite Perfect Hashing», *ACM Transactions on Database Systems,* volumen 14, número 2 (1989), páginas 231-263.

[Ramakrishnan et al. 1992a] R. Ramakrishnan, D. Srivastava y S. Sudarshan, *Controlling the Search in Bottom-up Evaluation* (1992).

[Ramakrishnan et al. 1992b] R. Ramakrishnan, D. Srivastava y S. Sudarshan, «CORAL: Control, Relations, and Logic», *Proc. of the International Conf. on Very Large Databases* (1992), páginas 238-250.

[Ramakrishnan et al. 1992c] R. Ramakrishnan, D. Srivastava y S. Sudarshan. «Efficient Bottomup Evaluation of Logic Programs», en *Vandewalle* [*1992*] (1992).

[Ramakrishnan et al. 1993] R. Ramakrishnan, D. Srivastava, S. Sudarshan y P. Sheshadri, «Implementation of the CORAL Deductive Database System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993), páginas 167-176.

[Ramakrishnan y Gehrke 2000] R. Ramakrishnan y J. Gehrke, *Database Management Systems*, 2.ª edición, McGraw Hill (2000).

[Ramakrishnan y Ullman 1995] R. Ramakrishnan y J. D. Ullman, «A Survey of Deductive Database Systems», *Journal of Logic Programming,* volumen 23, número 2 (1995), páginas 125-149.

[Ramesh et al. 1989] R. Ramesh, A. J. G. Babu y J. P. Kincaid, «Index Optimization: Theory and Experimental Results», *ACM Transactions on Database Systems,* volumen 14, número 1 (1989), páginas 41-74.

[Rangan et al. 1992] P. V. Rangan, H. M. Vin y S. Ramanathan, «Designing an On-Demand Multimedia Service», *Communications Magazine,* volumen 1, número 1 (1992), páginas 56-64.

[Reason et al. 1996] J. M. Reason, L. C. Yun, A. Y. Lao y D. G. Messerschmitt. «Asynchronous Video: Coordinated Video Coding and Transport for Heterogeneous Networks with Wireless Access», en *Imielinski and Korth [1996] Capítulo 10* (1996).

[Reed 1978] D. Reed, *Naming and Synchronization in a Decentralized Computer System*. Tesis doctoral, Department of Electrical Engineering, MIT, Cambridge (1978).

[Reed 1983] D. Reed, «Implementing Atomic Actions on Decentralized Data», *Transactions on Computer Systems*, volumen 1, número 1 (1983), páginas 3-23.

[Reuter 1989] A. Reuter, «ConTracts: A Means for Extending Control Beyond Transaction Boundaries», *Proc. of the 3rd International Workshop on High Performance Transaction Systems* (1989).

[Richardson et al. 1987] J. Richardson, H. Lu y K. Mikkilineni, «Design and Evaluation of Parallel Pipelined Join Algorithms», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987).

[Ries y Stonebraker 1977] D. R. Ries y M. Stonebraker, «Effects of Locking Granularity in a Database Management System», *ACM Transactions on Database Systems,* volumen 2, número 3 (1977), páginas 233-246.

[Rivest 1976] R. L. Rivest, «Partial Match Retrieval Via the Method of Superimposed Codes», *SIAM Journal of Computing,* volumen 5, número 1 (1976), páginas 19-50.

[Rivest et al. 1978] R. L. Rivest, A. Shamir y L. Adelman, «On Digital Signatures and Public Key Cryptosystems», *Communications of the ACM,* volumen 21, número 2 (1978), páginas 120-126.

[Robinson 1981] J. Robinson, «The k-d-B Tree: A Search Structure for Large Multidimensional Indexes», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1981), páginas 10-18.

[Rosch y Wethington 1999] W. L. Rosch y A.Wethington, *TheWinn L. Rosch Hardware Bible*, 5.ª edición, Que (1999).

[Rosenblum y Ousterhout 1991] M. Rosenblum y J. K. Ousterhout, «The Design and Implementation of a Log-Structured File System», *Proc. of the International Conf. on Architectural Support for Programming Languages and Operating Systems* (1991), páginas 1-15.

[Rosenkrantz et al. 1978] D. J. Rosenkrantz, R. E. Stearns y P. M. L. II, «System Level Concurrency Control For Distributed Data Base Systems», *ACM Transactions on Database Systems*, volumen 3, número 2 (1978), páginas 178-198.

[Rosenthal y Reiner 1984] A. Rosenthal y D. Reiner, «Extending the Algebraic Framework of Query Processing to Handle Outerjoins», *Proc. of the International Conf. on Very Large Databases* (1984), páginas 334-343.

[Ross 1990] K. A. Ross. «Modular Stratification and Magic Sets for DATALOG Programs with Negation», *Proc. of theACM SIGMOD Conf. on Management of Data* (1990).

[Ross 1999] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists*, Harcourt / Academic Press (1999).

[Ross et al. 1996] K. Ross, D. Srivastava y S. Sudarshan, «Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Ross y Srivastava 1997] K. A. Ross y D. Srivastava, «Fast Computation of Sparse Datacubes», en *Proc. of the International Conf. on Very Large Databases* (1997), páginas 116-125.

[Roth et al. 1988] M. A. Roth, H. F. Korth y A. Silberschatz, «Extended Algebra and Calculus for Nested Relational Databases», *ACM Transactions on Database Systems*, volumen 13, número 4 (1988), páginas 389-417.

[Roth et al. 1989] M. A. Roth, H. F. Korth y A. Silberschatz, «Null Values in Nested Relational Databases», *Acta Informatica,* volumen 26 (1989), páginas 615-642.

[Roth y Korth 1987] M. A. Roth y H. F. Korth, «The Design of ¬1NF Relational Databases into Nested Normal Form», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987), páginas 143-159.

[Rothermel y Mohan 1989] K. Rothermel y C. Mohan, «ARIES/NT: A Recovery Method Based on Write-Ahead Logging for Nested Transactions», *Proc. of the International Conf. on Very Large Databases* (1989), páginas 337-346.

[Rothnie et al. 1977] J. B. Rothnie, Jr. y N. Goodman, «A Survey of Research and Development in Distributed Database Management», *Proc. of the International Conf. on Very Large Databases* (1977), páginas 48-62.

[Roy et al. 2000] P. Roy, S. Seshadri, S. Sudarshan y S. Bhobhe, «Efficient and Extensible Algorithms for Multi-Query Optimization», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2000).

[Ruemmler y Wilkes 1994] C. Ruemmler y J. Wilkes, «An Introduction to Disk Drive Modeling», *IEEE Computer*, volumen 27, número 3 (1994), páginas 17-27.

[Rusinkiewicz y Sheth 1995] M. Rusinkiewicz y A. Sheth. «Specification and Execution of Transactional Workflows», en *Kim [1995]*, páginas 592-620 (1995).

[Rustin 1972] R. Rustin, *Data Base Systems*, Prentice Hall (1972).

[Rys 2001] M. Rys, «Bringing the Internet to your Database: Using SQL Server 2000 and XML to build Web and B2B Applications», *Proc. of the International Conf. on Data Engineering* (2001).

[Sagiv y Yannakakis 1981] Y. Sagiv y M. Yannakakis, «Equivalence among Relational Expressions with the Union and Difference Operators», *Proc. of the ACM SIGMOD Conf. on Management of Data*, volumen 27, número 4 (1981).

[Sahuguet 2001] A. Sahuguet, «Kweelt: More than just "yet another framework to query XML"!», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2001).

[Salem et al. 1994] K. Salem, H. García-Molina y J. Sands, «Altruistic Locking», *ACM Transactions on Database Systems*, volumen 19, número 1 (1994), páginas 117-165.

[Salem y García-Molina 1986] K. Salem y H. García-Molina, «Disk Striping», *Proc. of the International Conf. on Data Engineering* (1986), páginas 336-342.

[Salton 1989] G. Salton, *Automatic Text Processing*, Addison Wesley (1989).

[Samet 1990] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison Wesley (1990).

[Samet 1995a] H. Samet, «General Research Issues in Multimedia Database Systems», *ACM Computing Survey*, volumen 27, número 4 (1995), páginas 630-632.

[Samet 1995b] H. Samet. «Spatial Data Structures», en *Kim [1995],* páginas 361-385 (1995).

[Samet y Aref 1995] H. Samet y W. Aref. «Spatial Data Models and Query Processing», en *Kim [1995]*, páginas 338-360 (1995).

[Sanders 1998] R. E. Sanders, *ODBC 3.5 Developer's Guide*, McGraw Hill (1998).

[Sanders 2000] R. E. Sanders, *DB2 Universal Database SQL Developer's Guide*, McGraw Hill (2000).

[Sarawagi 2000] S. Sarawagi, «User-Adaptive Exploration of Multidimensional Data», en *Proc. of the International Conf. on Very Large Databases* (2000), páginas 307-316.

[Schlageter 1981] G. Schlageter, «Optimistic Methods for Concurrency Control in Distributed Database Systems», *Proc. of the International Conf. on Very Large Databases* (1981), páginas 125-130.

[Schmid y Swenson 1975] H. A. Schmid y J. R. Swenson, «On the Semantics of the Relational Model», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1975), páginas 211-223.

[Schneider 1982] H. J. Schneider, *Distributed Data Bases* (1982).

[Schneider y DeWitt 1989] D. Schneider y D. DeWitt, «A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1989).

[Schning 2001] H. Schning, «Tamino - A DBMS designed for XML», *Proc. of the International Conf. on Data Engineering* (2001), páginas 149-154.

[Selinger et al. 1979] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie y T. G. Price, «Access Path Selection in a Relational Database System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1979), páginas 23-34.

[Selinger y Adiba 1980] P. G. Selinger y M. E. Adiba, «Access Path Selection in Distributed Database Management Systems», informe técnico RJ2338, IBM Research Laboratory, San José (1980).

[Sellis 1988] T. K. Sellis, «Multiple Query Optimization», *ACM Transactions on Database Systems*, volumen 13, número 1 (1988), páginas 23-52.

[Sellis et al. 1987] T. K. Sellis, N. Roussopoulos y C. Faloutsos, «The R+-Tree: A Dynamic Index for Multi-Dimensional Objects», *Proc. of the International Conf. on Very Large Databases* (1987), páginas 507-518.

[Seshadri et al. 1996] P. Seshadri, H. Pirahesh y T. Y. C. Leung, «Complex Query Decorrelation», *Proc. of the International Conf. on Data Engineering* (1996), páginas 450-458.

[Seshadri y Naughton 1992] S. Seshadri y J. Naughton, «Sampling Issues in Parallel Database Systems», *Proc. of the International Conf. on Extending Database Technology* (1992).

[Sha et al. 1988] L. Sha, J. Lehoczky y D. Jensen, «Modular Concurrency Control and Failure Recovery», *IEEE Transactions on Computing,* volumen 37, número 2 (1988), páginas 146-159.

[Shafer et al. 1996] J. C. Shafer, R. Agrawal y M. Mehta, «SPRINT: A Scalable Parallel Classifier for Data Mining», *Proc. of the International Conf. on Very Large Databases* (1996), páginas 544-555.

[Shanmugasundaram et al. 1999] J. Shanmugasundaram, G. He, K. Tufte, C. Zhang, D. DeWitt y J. Naughton, «Relational Databases for Querying XML Documents: Limitations and Opportunities», *Proc. of the International Conf. on Very Large Databases* (1999).

[Shanmugasundaram et al. 2000] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh y B. Reinwald, «Relational Databases for Querying

XML Documents: Limitations and Opportunities», *Proc. of the International Conf. on Very Large Databases* (2000), páginas 65-76.

[Shapiro 1986] L. D. Shapiro, «Join Processing in Database Systems with Large Main Memories», *ACM Transactions on Database Systems,* volumen 11, número 3 (1986), páginas 239-264.

[Shasha 1992] D. Shasha, *Database Tuning: A Principled Approach*, Prentice Hall (1992).

[Shasha et al. 1995] D. Shasha, F. Llirabat, E. Simon y P. Valduriez, «Transaction Chopping: Algorithms and Performance Studies», *ACM Transactions on Database Systems*, volumen 20, número 3 (1995), páginas 325-363.

[Shasha y Goodman 1988] D. Shasha y N. Goodman, «Concurrent Search Structure Algorithms», *ACM Transactions on Database Systems*, volumen 13, número 1 (1988), páginas 53-90.

[Shatdal y Naughton 1993] A. Shatdal y J. Naughton, «Using Shared Virtual Memory for Parallel Join Processing», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993).

[Sheard y Stemple 1989] T. Sheard y D. Stemple, «Automatic Verification of Database Transaction Safety», *ACM Transactions on Database Systems,* volumen 14, número 3 (1989), páginas 322-368.

[Sibley 1976] E. Sibley, «The Development of Database Technology», *ACM Computing Survey*, volumen 8, número 1 (1976), páginas 1-5.

[Signore et al. 1995] R. Signore, J. Creamer y M. O. Stegman, *The ODBC Solution: Open Database Connectivity Distributed Environments*, McGraw Hill (1995).

[Silberschatz 1982] A. Silberschatz, «A Multiversion Concurrency Control Scheme With No Rollbacks», *Proc. of the ACM Symposium on Principles of Distributed Computing* (1982), páginas 216-223.

[Silberschatz et al. 1990] A. Silberschatz, M. R. Stonebraker y J. D. Ullman, «Database Systems: Achievements and Opportunities», *ACM SIGMOD Record,* volumen 19, número 4 (1990).

[Silberschatz et al. 1996] A. Silberschatz, M. Stonebraker y J. Ullman, «Database Research: Achievements and Opportunities into the 21st Century», informe técnico CS-TR-96-1563, Department of Computer Science, Stanford University, Stanford (1996).

[Silberschatz y Galvin 1998] A. Silberschatz y P. Galvin, *Operating System Concepts*, 5.ª edición, John Wiley and Sons (1998).

[Silberschatz y Kedem 1980] A. Silberschatz y Z. Kedem, «Consistency in Hierarchical Database Systems», *Journal of the ACM,* volumen 27, número 1 (1980), páginas 72-80.

[Simmen et al. 1996] D. Simmen, E. Shekita y T. Malkemus, «Fundamental Techniques for Order Optimization», *Proc. of the ACM SIGMOD Conf. on Management of Data*, Montreal, Canada (1996), páginas 57-67.

[Simmons 1979] G. J. Simmons, «Symmetric and Asymmetric Encryption», *ACM Computing Survey*, volumen 11, número 4 (1979), páginas 304-330.

[Skarra y Zdonik 1986] A. Skarra y S. Zdonik, «The Management of Changing Types in an Object-Oriented Database», *Proc. of the Object-Oriented Programming Languages, Systems and Applications Conf. (OOPSLA)* (1986).

[Skarra y Zdonik 1989] A. Skarra y S. Zdonik. «Concurrency Control in Object-Oriented Databases», en *Kim and Lochovsky [1989]*, páginas 395-421 (1989).

[Skeen 1981] D. Skeen, «Non-blocking Commit Protocols», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1981), páginas 133-142.

[Smith y Smith 1977] J. M. Smith y D. C. P. Smith, «Database Abstractions: Aggregation and Generalization», *ACM Transactions on Database Systems,* volumen 2, número 2 (1977), páginas 105-133.

[Snodgrass 1987] R. Snodgrass, «The Temporal Query Language TQuel», *ACM Transactions on Database Systems*, volumen 12, número 2 (1987), páginas 247-298.

[Snodgrass et al. 1994] R. Snodgrass et al., «TSQL2 Language Specification», *ACM SIGMOD Record*, volumen 23, número 1 (1994), páginas 65-86.

[Snodgrass y Ahn 1985] R. Snodgrass e I. Ahn, «A Taxonomy of Time in Databases,», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1985), páginas 236-246.

[Soo 1991] M. Soo, «Bibliography on Temporal Databases», *ACM SIGMOD Record,* volumen 20, número 1 (1991), páginas 14-23.

[Soparkar et al. 1991] N. Soparkar, H. F. Korth y A. Silberschatz, «Failure-Resilient Transaction Management in Multidatabases», *IEEE Computer,* volumen 24, número 12 (1991), páginas 28-36.

[Soparkar et al. 1995] N. Soparkar, H. F. Korth y A. Silberschatz, «Databases with Deadline and Contingency Constraints», *IEEE Transactions on Knowledge and Data Engineering*, volumen 7, número 4 (1995), páginas 552-565.

[Spector y Schwarz 1983] A. Z. Spector y P. M. Schwarz, «Transactions: A Construct for Reliable Distributed Computing», *Operating Systems Review,* volumen 17, número 2 (1983), páginas 18-35.

[Srikant y Agrawal 1996a] R. Srikant y R. Agrawal, «Mining Quantitative Association Rules in Large Relational Tables», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Srikant y Agrawal 1996b] R. Srikant y R. Agrawal, «Mining Sequential Patterns: Generalizations and Performance Improvements», *Proc. of the International Conf. on Extending Database Technology* (1996), páginas 3-17.

[Srinivasan et al. 2000a] J. Srinivasan, S. Das, C. Freiwald, E. I. Chong, M. Jagannath, A. Yalamanchi, R. Krishnan, A.-T. Tran, S. DeFazio y J. Banerjee, «Oracle8i Index-Organized Table and Its Application to New Domains», *Proc. of the International Conf. on Very Large Databases* (2000), páginas 285-296.

[Srinivasan et al. 2000b] J. Srinivasan, R. Murthy, S. Sundara, N. Agarwal y S. DeFazio, «Extensible Indexing: A Framework for Integrating Domain-Specific Indexing Schemes into Oracle8i», *Proc. of the International Conf. on Data Engineering* (2000), páginas 91-100.

[Srivastava et al. 1995] D. Srivastava, S. Sudarshan, R. Ramakrishnan y J. Naughton, «Space Optimization in Deductive Databases», *ACM Transactions on Database Systems*, volumen 20, número 4 (1995), páginas 472-516.

[Stachour y Thuraisingham 1990] P. D. Stachour y B. Thuraisingham, «Design of LDV: A Multilevel Secure Relational Database Management System», *IEEE Transactions on Knowledge and Data Engineering*, volumen 2, número 2 (1990), páginas 190-209.

[Stallings 1998] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 2.ª edición, Prentice Hall (1998).

[Stam y Snodgrass 1988] R. Stam y R. Snodgrass, «A Bibliography on Temporal Databases», *IEEE Transactions on Knowledge and Data Engineering*, volumen 7, número 4 (1988), páginas 231-239.

[Stefik y Bobrow 1986] M. Stefik y D. G. Bobrow, «Object-Oriented Programming: Themes and Variations», *The AI Magazine* (1986), páginas 40-62.

[Stone 1993] H. S. Stone, *High-Performance Computer Architecture*, 3.ª edición, Addison Wesley (1993).

[Stonebraker 1975] M. Stonebraker, «Implementation of Integrity Constraints and Views by Query Modification», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1975), páginas 65-78.

[Stonebraker 1980] M. Stonebraker, «Retrospection on a Database System», *ACM Transactions on Database Systems*, volumen 5, número 2, También aparece en Stonebraker [1986b], páginas 46-62 (1980), páginas 225-240.

[Stonebraker 1981] M. Stonebraker, «Operating System Support for Database Management», *Communications of the ACM*, volumen 24, número 7, También aparece en Stonebraker [1986b], páginas 172-182 (1981), páginas 412-418.

[Stonebraker 1986a] M. Stonebraker, «Inclusion of New Types in Relational Database Systems», *Proc. of the International Conf. on Data Engineering* (1986), páginas 262-269.

[Stonebraker 1986b] M. Stonebraker, editor, *The Ingres Papers*, Addison Wesley (1986).

[Stonebraker et al. 1976] M. Stonebraker, E. Wong, P. Kreps y G. D. Held, «The Design and Implementation of INGRES», *ACM Transactions on Database Systems,* volumen 1, número 3 (1976), páginas 189-222.

[Stonebraker y Hellerstein 1998] M. Stonebraker y J. Hellerstein, *Readings in Database Systems*, 3.ª edición, Morgan Kaufmann (1998).

[Stonebraker y Rowe 1986] M. Stonebraker y L. Rowe, «The Design of POSTGRES», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1986).

[Stonebraker y Wong 1974] M. Stonebraker y E. Wong, «Access Control in a Relational Database Management System by Query Modification», *Proc. of the ACM National Conference* (1974), páginas 180-187.

[Stonebraker et al. 1989] M. Stonebraker, P. Aoki y M. Seltzer, «Parallelism in XPRS», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1989).

[Stroustrup 1988] B. Stroustrup, *What Is Object-Oriented Programming?* (1988).

[Stroustrup 1997] B. Stroustrup, *The C++ Programming Language*, 3.ª edición, Addison Wesley (1997).

[Stuart et al. 1984] D. G. Stuart, G. Buckley y A. Silberschatz, «A Centralized Deadlock Detection Algorithm», informe técnico, Department of Computer Sciences, University of Texas, Austin (1984).

[Sudarshan y Ramakrishnan 1991] S. Sudarshan y R. Ramakrishnan, «Aggregation and Relevance in Deductive Databases», *Proc. of the International Conf. on Very Large Databases* (1991).

[Swami y Gupta 1988] A. Swami y A. Gupta, «Optimization of Large Join Queries», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1988), páginas 8-17.

[Tanenbaum 1996] A. S. Tanenbaum, *Computer Networks*, 3.ª edición, Prentice Hall (1996).

[Tansel et al. 1993] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev y R. Snodgrass, *Temporal Databases: Theory, Design and Implementation*, Benjamin Cummings, Redwood City (1993).

[Tendick y Matloff 1994] P. Tendick y N. Matloff, «A Modified Random Perturbation Method for Database Security», *ACM Transactions on Database Systems,* volumen 19, número 1 (1994), páginas 47-63.

[Teorey et al. 1986] T. J. Teorey, D. Yang y J. P. Fry, «A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model», *ACM Computing Survey*, volumen 18, número 2 (1986), páginas 197-222.

[Thalheim 2000] B. Thalheim, *Entity-Relationship Modeling*: *Foundations of Database Technology*, Springer Verlag (2000).

[Thomas 1979] R. H. Thomas, «A Majority Consensus Approach to Concurrency Control», *ACM Transactions on Database Systems*, volumen 4, número 2 (1979), páginas 180-219.

[Thomas 1996] S. A. Thomas, *IPng and the TCP/IP Protocols: Implementing the Next Generation Internet*, John Wiley and Sons (1996).

[Todd 1976] S. J. P. Todd, «The Peterlee Relational Test Vehicle - A System Overview», *IBM Systems Journal,* volumen 15, número 4 (1976), páginas 285-308.

[Traiger et al. 1982] I. L. Traiger, J. N. Gray, C. A. Galtieri y B. G. Lindsay, «Transactions and Consistency in Distributed Database Management Systems», *ACM Transactions on Database Systems,* volumen 7, número 3 (1982), páginas 323-342.

[Tsou y Fischer 1982] D. M. Tsou y P. Fischer, «Decomposition of a Relation Scheme into Boyce-Codd Normal Form», *ACM SIGACT News,* volumen 14, número 3 (1982), páginas 23-29.

[Tsukuda et al. 1992] S. Tsukuda, M. Nakano, M. Kitsuregawa y M. Takagi, «Parallel Hash Join on Shared-Everything Multiprocessor», *Proc. of the International Conf. on Data Engineering* (1992).

[Tsur y Zaniolo 1986] S. Tsur y C. Zaniolo, «LDL: A Logic-Based Data-Language», *Proc. of the International Conf. on Very Large Databases* (1986), páginas 33-41.

[Tuzhilin y Clifford 1990] A. Tuzhilin y J. Clifford, «A Temporal Relational Algebra as a Basis for Temporal Relational Completeness», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 13-23.

[Ullman 1988] J. D. Ullman, *Principles of Database and Knowledge-base Systems*, volumen 1, Computer Science Press, Rockville (1988).

[Ullman 1989] J. D. Ullman, *Principles of Database and Knowledge-base Systems*, volumen 2, Computer Science Press, Rockville (1989).

[Umar 1997] A. Umar, *Application (Re)Engineering*: *Building Web-Based Applications and Dealing With Legacies*, Prentice Hall (1997).

[Unger et al. 1984] E. A. Unger, P. S. Fisher y J. Slonim, *Advances in Data Base Management*, volumen 2, John Wiley and Sons (1984).

[UniSQL 1991] *UniSQL/X Database Management System User's Manual: Release 1.2*. UniSQL Inc. (1991).

[US Dept. of Commerce 1977] *United States Department of Commerce, Data Encryption Standard*. National Bureau of Standards (1977).

[US Dept. of Defense 1985] *Department of Defense Trusted Computer System Evaluation Criteria*. National Computer Security Center (1985).

[Vandewalle 1992] J. Vandewalle, *The State of the Art in Computer Systems and Software Engineering*, Kluwer Academic (1992).

[Verhofstad 1978] J. S. M. Verhofstad, «Recovery Techniques for Database Systems», *ACM Computing Survey*, volumen 10, número 2 (1978), páginas 167-195.

[Vista 1998] D. Vista, «Integration of Incremental View Maintenance into Query Optimizers», *Proc. of the International Conf. on Extending Database Technology* (1998).

[Wachter y Reuter 1992] H. Wachter y A. Reuter. «The ConTract Model», en A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*. Morgan Kaufmann (1992).

[Walton et al. 1991] C.Walton, A. Dale y R. Jenevein, «A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins», *Proc. of the International Conf. on Very Large Databases* (1991).

[Weihl y Liskov 1990] W. Weihl y B. Liskov, «Implementation of Resilient, Atomic Data Types», en *Zdonik and Maier [1990]*, páginas 332-344 (1990).

[Weikum 1991] G.Weikum, «Principles and Realization Strategies of Multilevel Transaction Management», *ACM Transactions on Database Systems,* volumen 16, número 1 (1991).

[Weikum et al. 1990] G. Weikum, C. Hasse, P. Broessler y P. Muth, «Multi-Level Recovery», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990), páginas 109-123.

[Weikum y Schek 1984] G. Weikum y H. J. Schek, «Architectural Issues of Transaction Management in Multi-Level Systems», *Proc. of the International Conf. on Very Large Databases* (1984), páginas 454-465.

[Weltman y Dahbura 2000] R. Weltman y T. Dahbura, *LDAP Programming with Java*, Addison Wesley (2000).

[Whang y Krishnamurthy 1990] K. Whang y R. Krishnamurthy, «Query Optimization in a Memory-Resident Domain Relational Calculus Database System», *ACM Transactions on Database Systems*, volumen 15, número 1 (1990), páginas 67-95.

[White y DeWitt 1992] S. J. White y D. J. DeWitt, «A Performance Study of Alternative Object Faulting and Pointer Swizzling Strategies», *Proc. of the International Conf. on Very Large Databases* (1992).

[White y DeWitt 1994] S. J. White y D. J. DeWitt, «QuickStore: A High PerformanceMapped Object Store», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1994).

[Widom et al. 1991] J. Widom, R. Cochrane y B. Lindsay, «Implementing Set-Oriented Production Rules as an Extension to Starburst», *Proc. of the International Conf. on Very Large Databases* (1991).

[Widom y Finkelstein 1990] J. Widom y S. Finkelstein, «Set-Oriented Production Rules in Relational Database Systems», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990).

[Wilkinson et al. 1990] K. Wilkinson, P. Lyngbaek y W. Hasan, «The Iris Architecture and Implementation», *IEEE Transactions on Knowledge and Data Engineering* (1990), páginas 63-75.

[Wilschut et al. 1995] A. N. Wilschut, J. Flokstra y P. M. Apers, «Parallel Evaluation of MultiJoin Queues», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 115-126.

[Wilson 1990] P. R. Wilson, «Pointer Swizzling at Page Fault Time: Efficiently Supporting Huge Address Spaces on Standard Hardware», informe técnico UIC-EECS-90-6, University of Illinois at Chicago (1990).

[Winslett et al. 1994] M. Winslett, K. Smith y X. Qian, «Formal Query Languages for Secure Relational Databases», *ACM Transactions on Database Systems,* volumen 19, número 4 (1994), páginas 626-662.

[Wipfler 1987] A. J. Wipfler, *CICS: Application Development and Programming*, Macmillan Publishing, Nueva York (1987).

[Witten et al. 1999] I. H. Witten, A. Moffat y T. C. Bell, *Managing Gigabytes: compressing and indexing documents and images*, Morgan Kaufmann (1999).

[Witten y Frank 1999] I. H. Witten y E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann (1999).

[Wolf 1991] J. Wolf, «An Effective Algorithm for Parallelizing Hash Joins in the Presence of Data Skew», *Proc. of the International Conf. on Data Engineering* (1991).

[Wong 1977] E. Wong, «Retrieving Dispersed Data from SDD-1: A System for Distributed Databases», *Proc. of the Berkeley Workshop on Distributed Data Management and Computer Networks* (1977), páginas 217-235.

[Wong 1983] E. Wong, «Dynamic Rematerialization- Processing Distributed Queries Using Redundant Data», *IEEE Transactions on Software Engineering,* volumen SE-9, número 3 (1983), páginas 228-232.

[Wu y Buchmann 1998] M. Wu y A. Buchmann, «Encoded Bitmap Indexing for Data Warehouses», en *Proc. of the International Conf. on Data Engineering* (1998).

[X/Open 1991] *X/Open Snapshot: X/Open DTP: XA Interface*. X/Open Company, Ltd. (1991).

[X/Open 1993] *X/Open Data Management: SQL Call Level Interface (CLI)*. X/Open Company, Ltd. (1993).

[Yan y Larson 1995] W. P. Yan y P. A. Larson, «Eager Aggregation and Lazy Aggregation», en *Proc. of the International Conf. on Very Large Databases*, Zurich (1995).

[Yannakakis 1981] M. Yannakakis, «Issues of Correctness in Database Concurrency Control by Locking», *Proc. of the IEEE Symposium on the Foundations of Computer Science* (1981), páginas 363-367.

[Yannakakis et al. 1979] M. Yannakakis, C. H. Papadimitriou y H. T. Kung, «Locking Protocols: Safety and Freedom from Deadlock», *Proc. of the IEEE Symposium on the Foundations of Computer Science* (1979), páginas 286-297.

[Zaniolo 1976] C. Zaniolo, *Analysis and Design of Relational Schemata for Database Systems*. Tesis doctoral, Department of Computer Science, University of California, Los Ángeles (1976).

[Zaniolo 1983] C. Zaniolo, «The Database Language GEM», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1983), páginas 207-218.

[Zaniolo et al. 1997] C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, R. Zicari y V. S. Subrahmanian, *Advanced Database Systems*, Morgan Kaufmann (1997).

[Zdonik y Maier 1990] S. Zdonik y D. Maier, *Readings in Object-Oriented Database Systems*, Morgan Kaufmann (1990).

[Zeller y Gray 1990] H. Zeller y J. Gray, «An Adaptive Hash Join Algorithm for Multiuser Environments», *Proc. of the*

International Conf. on Very Large Databases (1990), páginas 186-197.

[Zhang et al. 1996] T. Zhang, R. Ramakrishnan y M. Livny, «BIRCH: An Efficient Data Clustering Method for Very Large Databases», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 103-114.

[Zhang y Mendelzon 1983] Z. Q. Zhang y A. O. Mendelzon. «A Graphical Query Language for Entity-Relationship Databases», en *Davis et al*. [*1983*], páginas 441-448 (1983).

[Zhuge et al. 1995] Y. Zhuge, H. García-Molina, J. Hammer y J.Widom, «View Maintenance in a Warehousing Environment», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 316-327.

[Zikopoulos et al. 2000] P. Zikopoulos, R. Melnyk y L. Logomirski, *DB2 fof Dummies*, Hungry Minds Inc. (2000).

[Zloof 1977] M. M. Zloof, «Query-by-Example: A Data Base Language», *IBM Systems Journal*, volumen 16, número 4 (1977), páginas 324-343.



DICCIONARIO BILINGÜE

N. del T.: En este diccionario bilingüe se recogen los términos más importantes que aparecen en el libro junto al correspondiente vocablo español para aclarar la traducción realizada, dado que no hay un consenso general. Esperamos que sirva de ayuda al lector en la identificación tanto de los términos anglosajones como de los españoles.

Notas:

- El símbolo / indica polisemia.
- Los paréntesis se usan para aclarar el contexto, o para la expansión y contracción de un acrónimo.
- *n*. indica un nombre.
- *v*. indica un verbo.
- *m*. indica género masculino.
- *f*. indica género femenino.

Inglés	Español
1NF (First normal form)	Primera forma normal (1FN)
2NF (Second normal form)	Segunda forma normal (2FN)
2PC protocol	Protocolo de compromiso de dos fases (C2F)
3NF (Third normal form)	Tercera forma normal (3FN)
3PC protocol	Protocolo de compromiso de tres fases (C3F)
4GL (Fourth-generation language)	Lenguaje de cuarta generación (L4G)
4NF (Fourth normal form)	Cuarta forma normal (4FN)
Aborted	Abortada (transacción)
Aborted acceptable termination state	Estado de terminación aceptable abortado
Aborted termination state	Estado de terminación abortado
Abstract	Abstracto
Abstract data	Datos abstractos
Acceptable termination state	Estado de terminación aceptable
Access paths	Caminos de acceso
Access time	Tiempo de acceso
Access-plan-selection	Selección del plan de acceso
ACID properties	Propiedades ACID
Active documents	Documentos activos
Active rules	Reglas activas
Acyclic	Acíclico
Aggregation	Agregación
Aggregation function	Función de agregación
Aggregation operation	Operación de agregación

Aggregation operator	Operador de agregación
Alerting	Advertencia
Alias	Alias
Aliases	Alias
Almaden Research Center	Centro de investigación de Almadén
American National Standards Institute (ANSI)	Instituto de Normalización Nacional Americano
ANSI (American National Standards Institute)	Instituto de Normalización Nacional Americano
Anticipatory standard	Norma anticipativa
Any	Any
API (Application programming interface)	Interfaz de programación de aplicaciones
Append operation	Operación append
Application programming interface (API)	Interfaz de programación de aplicaciones
Application programs	Programas de aplicación
Archival dump	Volcado de archivo
Area	Zona
Armstrong's axioms	Axiomas de Armstrong
Arpanet	Arpanet
Array	*Array*
Ascending order	Ascending order
Assertion	Aserto
Assignment	Asignación
Assignment operation	Operación asignación
Association rules	Reglas de asociación
Association rules discovery	Descubrimiento de reglas de asociación
Assume	Asumir
Asymmetric fragment and replicate	Fragmentos y réplicas asimétricos
Asynchronous transfer mode (ATM)	Modo de transferencia asíncrono (MTA)
ATM (Asynchronous transfer mode)	Modo de transferencia asíncrono (MTA)
Atom	Átomo
Atomic	Atómico
Atomic domain	Dominio atómico
Atomicity	Atomicidad
Attribute	Atributo
Attribute inheritance	Herencia de atributos
Attribute list	Lista de atributos
Attribute value skew	Sesgo de los valores de los atributos
Attribute-defined	Definido por atributo
Attribute-defined design constraints	Restricciones de diseño definidas por los atributos
Attribute-value skew	Sesgo de los valores
Audio and video data	Datos de sonido y de vídeo
Augmentation rule	Regla de la aumentatividad
Authorization	Autorización
Authorization graph	Grafo de autorización

Authorization restrictions	Restricciones de autorización
Average latency time	Tiempo medio de latencia
Average response time	Tiempo medio de respuesta
Average seek time	Tiempo medio de búsqueda
Avoidance	Evitación
Axiom	Axioma
B^+-tree	Árbol B^+
B^+-tree concurrency control algorithms	Algoritmos de control de concurrencia en árboles B^+
B^+-tree concurrency-control algorithm	Algoritmo de control de concurrencia en árboles B^+
B^+-tree file organization	Organización de archivos con árboles B^+
B^+-tree index	Índice de árbol B^+
B^+-tree index structure	Estructura de índice de árbol B^+
Back-end	Sistema subyacente
Backup coordinator	Coordinador copia de seguridad
Balanced	Equilibrado
Band join	Reunión de banda
Base	Base
Batch scaleup	Ampliabilidad por lotes
BCNF (Boyce-Codd normal form)	Forma normal de Boyce-Codd (FNBC)
Biased protocol	Protocolo sesgado
Big-endian form	Orden más significativo
Binary	Binario
Binary large object (blob)	Objeto en binario de gran tamaño
Binary search	Búsqueda binaria
Binary trees	Árboles binarios
Bitemporal relation	Relación bitemporal
Bit-interleaved parity organization	Organización de paridad con bits entrelazados
Bit-level striping	Distribución en el nivel de bit
Blind writes	Escrituras a ciegas
B-link trees	Árboles B enlazados
Blob (Binary large object)	Objeto en binario de gran tamaño
Block	Bloque
Block number	Número de bloque
Blocking	Bloqueo
Blocking problem	Problema de bloqueo
Block-interleaved parity organization	Organización de paridad con bloques entrelazados
Block-level striping	Distribución en el nivel de bloque
Block-replacement strategy	Estrategia para la sustitución de bloques
Body	Cuerpo
Bottleneck	Cuello de botella
Bottom-up	Ascendente
Bound variable	Variable ligada

Bounding box	Caja límite
Box	Caja
Boyce-Codd normal form (BCNF)	Forma normal de Boyce-Codd (FNBC)
Browsing	Exploración
B-tree index	Índice de árbol B
Bucket	Cajón
Bucket overflow	Desbordamiento de cajones
Bucket skew	Atasco de cajones
Buffer	Memoria intermedia
Buffer block	Bloque de memoria intermedia
Buffer management	Gestión de la memoria intermedia
Buffer manager	Gestor de la memoria intermedia
Build	Construir
Build input	Entrada para construir
Bully algorithm	Algoritmo luchador
Bus	Bus
By operation	Operación by
Byte code	Código intermedio
Byte-string representation	Representación en cadenas de bytes
Cache	Caché
Cache coherency	Coherencia de caché
Cache-coherency problem	Problema de coherencia caché
CAD (Computer-aided design)	Diseño asistido por computadora
Call back	Comunicar
Call-level interface (CLI)	Interfaz del nivel de llamadas
Candidate key	Clave candidata
Canonical cover	Recubrimiento canónico
Cartesian product	Producto cartesiano
Cartesian-product operation	Operación producto cartesiano
Cascadeless schedules	Planificaciones sin cascada
Cascading of the revoke	Retirada en cadena
Cascading rollback	Retroceso en cascada
CASE (Computer-aided software engineering)	Ingeniería del software asistida por computadora
CD-ROM (Compact-disk read-only memory)	Memoria sólo de lectura en disco compacto
Cell	Celda
Central processing unit (CPU)	Unidad central de procesamiento (UCP)
Centralized approach	Enfoque centralizado
Centralized databases	Bases de datos centralizadas
CICS monitor	Monitor CICS
Class	Clase
Class extents	Extensiones de clases
Classification	Clasificación

Classification hierarchy	Jerarquía de clasificación
Classification tree	Árbol de clasificación
CLI (Call-level interface)	Interfaz del nivel de llamadas
Client systems	Sistemas clientes
Client-server	Cliente-servidor
Client-server database systems	Sistemas de bases de datos cliente-servidor
Client-server interface protocol	Protocolo de interfaz cliente-servidor
Closed hashing	Asociación cerrada
Closed polygons	Polígonos cerrados
Closure	Cierre
Cluster (*n.*)	Agrupación
Cluster (*v.*)	Agrupar
Clustering	Agrupamiento
Clustering file organization	Organización de archivos en agrupaciones
Clustering index	Índice con agrupación
Coalesce	Fusionar
Coalescence	Fusión
Coalescence rule	Regla de la fusión
Coarse granularity	Granularidad gruesa
Coarse-granularity parallelism	Paralelismo de grano grueso
CODASYL DBTG 1991 databases	Bases de datos CODASYL DBTG 1991
Collection type	Tipo colección
Combine	Combinar
Combined tables	Tablas combinadas
Command	Orden
Commit	Comprometer
Commit log record	Registro de compromiso del registro histórico
Commit protocol	Protocolo de compromiso
Commited	Comprometido
Commited termination state	Estado de terminación comprometido
Commited transaction	Transacción comprometida
Commitment	Compromiso
Committed acceptable termination state	Estado de terminación aceptable comprometido
Common hash prefix	Prefijo común de la función de asociación
Common object request broker architecture (CORBA)	Arquitectura común de agente para peticiones de objetos
Common subexpressions	Subexpresiones comunes
Compact-disk read-only memory (CD-ROM)	Memoria sólo de lectura en disco compacto
Comparison	Comparación
Compatibility function	Función de compatibilidad
Compatible	Compatible
Compensating transaction	Transacción compensadora
Complementation rule	Regla de la complementariedad

Complete	Completo
Complete rules	Reglas completas
Completeness constraint	Restricción de completitud
Complex	Complejo
Complex interactive transaction	Transacción interactiva compleja
Complex object	Objeto complejo
Complex queries types	Consultas con tipos complejos
Complex types	Tipos complejos
Component	Componente (*m.*)
Composite	Compuesto
Composite attributes	Atributos compuestos
Composite index	Índice compuesto
Composite object	Objeto compuesto
Composition of relational operations	Composición de operaciones relacionales
Computed-aided design (CAD)	Diseño asistido por computadora
Computer application	Aplicación informática
Computer network	Red de computadoras
Computer Science Technical Report Index	Índice de informes técnicos de informática
Computer system	Sistema informático
Computer-aided software engineering (CASE)	Ingeniería del software asistida por computadora
Computer-aided-design (CAD) databases	Bases de datos para diseño asistido por computadora (CAD)
Conceptual	Conceptual
Conceptual level	Nivel conceptual
Conceptual-design	Diseño conceptual
Concurrency components	Componentes de concurrencia
Concurrency control	Control de la concurrencia
Concurrency control B^+-trees algorithm	Algoritmo para control de la concurrencia para los árboles B^+
Concurrency-control component	Componente de control de concurrencia
Concurrency-control manager	Gestor de control de concurrencia
Concurrency-control protocol	Protocolo de control de la concurrencia
Concurrency-control schemes	Esquemas de control de concurrencia
Condition box	Caja de condición
Condition-defined design constraints	Restricciones de diseño definidas por las condiciones
Conditions-defined constraints	Restricciones con condiciones definidas
Confidence	Confianza
Conflict equivalent	Equivalente en cuanto a conflictos
Conflict schedules	Planificaciones de conflictos
Conflict serializability	Secuencialidad en cuanto a conflictos
Conflict serializable	Secuenciable en cuanto a conflictos
Confluent	Confluente
Conjunctive selection	Selección conjuntiva
Conjunctive selection by intersection of identifiers	Selección conjuntiva mediante la intersección de identificadores

Conjunctive selection using composite index	Selección conjuntiva utilizando un índice compuesto
Conjunctive selection using one index	Selección conjuntiva utilizando un índice
Consistency	Consistencia
Consistency constraints	Restricciones de consistencia
Constraints completeness	Restricciones de completitud
Constructed graph	Grafo construido
Constructor	Constructora
Containment hierarchy	Jerarquía de continentes
Contains	Contiene
Context switch	Cambio de contexto
Contiguously	De manera contigua
Continuous connection	Conexión continua
Continuous-media data	Datos de medios continuos
Convenient	Práctico
Coordinator-failure protocol	Protocolo de fallo del coordinador
CORBA (Common object request broker architecture)	Arquitectura común de agente para peticiones de objetos
Cost of parallel evaluation of operations	Coste de la evaluación paralela de operaciones
Cost-based optimization	Optimización basada en el coste
Count aggregate function	Función de agregación count
CPU (Central processing unit)	Unidad central de procesamiento (UCP)
Crash	Caída
Crash-recovery subsystem	Subsistema de recuperación de caídas
Cross join	Reunión cruzada
Cross-tab	Tabla cruzada
Cross-tabulation	Tabla de entradas cruzadas
Current page table	Tabla de páginas actual
Cursor stability	Estabilidad del cursor
Checkin	Marcar
Checkout	Desmarcar
Checkpoint	Punto de revisión
Checksum	Comprobación de suma
Child (or member)	Hijo (o miembro)
DAG (Directed acyclic graph)	Grafo acíclico dirigido (GAD)
Dangling	Colgante
Dangling pointer	Puntero colgante
Data abstraction	Abstracción de datos
Data archival	Almacenamiento de datos
Data broadcast	Difusión de datos
Data caching	Almacenamiento de datos en caché
Data consistency	Consistencia de los datos
Data dictionary	Diccionario de datos
Data directory	Directorio de datos

Data encapsulating	Encapsulamiento de datos
Data encryption standard (DES)	Norma de cifrado de datos
Data fragmentation	Fragmentación de datos
Data inconsistency	Inconsistencia de datos
Data independence	Independencia de datos
Data integrity	Integridad de los datos
Data manipulation	Manipulación de datos
Data mining	Recopilación de datos
Data model	Modelo de datos
Data naming	Denominación de datos
Data parallelism	Paralelismo de datos
Data partitioning	División de datos
Data pollution	Contaminación de los datos
Data redundancy	Redundancia de datos
Data replication	Réplica de datos
Data storage	Almacenamiento de datos
Data striping	Distribución de datos
Data structure definition	Definición de las estructuras de datos
Data structure diagram	Diagrama de la estructura de datos
Data transfer failure	Fallo en la transferencia de datos
Data transmission cost	Coste de la transmisión de datos
Data visualization	Visualización de datos
Data warehouse	Almacén de datos
Database (DB)	Base de datos (BD)
Database active system	Sistema de bases de datos activas
Database administrator (DBA)	Administrador de la base de datos (ABD)
Database applications	Aplicaciones de bases de datos
Database buffer	Memoria intermedia de la base de datos
Database CAD	CAD de bases de datos
Database design	Diseño de bases de datos
Database graph	Grafo de la base de datos
Database instance	Ejemplar de la base de datos
Database integrity	Integridad de las bases de datos
Database management system (DBMS)	Sistema gestor de bases de datos (SGBD)
Database schema	Esquema de la base de datos
Database security	Seguridad de la base de datos
Database System Concepts	Fundamentos de bases de datos
Database Task Group	Grupo de trabajo sobre bases de datos
Database tree	Árbol de la base de datos
Database-standard specification	Especificación de una norma para las bases de datos
Data-cube operator	Operador data-cube
Data-definition language (DDL)	Lenguaje de definición de datos (LDD)

Data-item utilization	Utilización de elementos de datos
Datalog	Datalog
Datalog language fixpoint	Punto fijo del lenguaje Datalog
Datalog language recursion	Recursividad del lenguaje Datalog
Datalog language rules	Reglas del lenguaje Datalog
Datalog rules	Reglas de Datalog
Data-manipulation language (DML)	Lenguaje de manipulación de datos (LMD)
Data-server systems	Sistemas servidores de datos
Data-structure diagram	Diagrama de estructura de datos
Data-transfer rate	Velocidad de transferencia de datos
Data-visualization systems	Sistemas de visualización de datos
DB (Database)	Base de datos (BD)
DBA (Database administrator)	Administrador de bases de datos (ABD)
DBMS (Database management system)	Sistema gestor de bases de datos (SGBD)
DBTG set	Conjunto DBTG
DDL (Data-definition language)	Lenguaje de definición de datos (LDD)
De facto standard	Norma de facto
Deadline	Tiempo límite
Deadlock	Interbloqueo
Deadlock detection	Detección de interbloqueos
Deadlock handling	Tratamiento de interbloqueos
Deadlock prevention	Prevención de interbloqueos
Deadlock recovery	Recuperación de interbloqueos
Deadlock-detection algorithm	Algoritmo para la detección de interbloqueos
Deadlock-detention and deadlock-recovery scheme	Esquema de detección y recuperación de interbloqueos
Deadlock-handling algorithm	Algoritmo para el tratamiento de interbloqueos
Deadlock-prevention protocol	Protocolo de prevención de interbloqueos
Decision-support queries	Consultas de ayuda para las decisiones
Decision-support systems	Sistemas de ayuda para las decisiones
Decompose	Descomponer
Decomposition	Descomposición
Decomposition rule	Regla de la descomposición
Decoupled	Desacoplada
Decrypt	Descifrar
Deferred	Diferida
Deferred modification	Modificación diferida
Degree	Grado
Degree of relevance	Grado de relevancia
Degree-two consistency	Consistencia de grado dos
Delete	Borrar
Delete operation	Operación delete
Deletion	Borrado

Demand-driven pipelining	Encauzamiento bajo demanda
Dense index	Índice denso
Dependency graph	Grafo de dependencia
Dependency preservation	Conservación de las dependencias
Dependency-preserving decomposition	Descomposición con conservación de las dependencias
Dependency-preserving decomposition	Descomposición que conserva las dependencias
Dependent joins	Reuniones dependientes
Dereference	Desreferenciar
Dereferencing	Desreferenciación
Derived attributes	Atributos derivados
Derived relations	Relaciones derivadas
DES (Data encryption standard)	Norma de cifrado de datos
Descending order	Orden descendente
Descriptive attributes	Atributos descriptivos
Design constraints	Restricciones de diseño
Design databases	Bases de datos para diseño
Destructor	Destructor
Deswizzling	Devolución
Difference rule	Regla de la diferencia
Digital signature	Firma digital
Digital video disk (DVD)	Videodisco digital (VDD)
Direct-access storage	Almacenamiento de acceso directo
Directed acyclic graph (DAG)	Grafo acíclico dirigido (GAD)
Discriminator	Discriminante
Disjoint subtrees	Subárboles disjuntos
Disjointness constraint	Restricción sobre el carácter disjunto
Disjunctive selection	Selección disyuntiva
Disjunctive selection by union of identifiers	Selección disyuntiva mediante la unión de identificadores
Disk arm	Brazo del disco
Disk buffer	Memoria intermedia de disco
Disk controller	Controlador de disco
Disk failure	Fallo de disco
Disk mirroring	Discos imagen
Disk storage	Almacenamiento en disco
Disk-arm-scheduling	Planificación del brazo del disco
Display	Visualizar
Distributed data	Datos distribuidos
Distributed database systems	Sistemas distribuidos de bases de datos
Distributed environment	Entorno distribuido
Distributed Gopher information systems	Sistemas distribuidos de información Gopher
Distributed hypertext systems	Sistemas distribuidos de hipertexto
Distributed information systems	Sistemas distribuidos de información

Distributed Internet information systems	Sistemas distribuidos de información en Internet
Distributed query processing	Procesamiento distribuido de consultas
Distributed transaction model	Modelo de transacciones distribuidas
Distributed virtual-memory	Memoria virtual distribuida
Distributed virtual-memory architecture	Arquitecturas distribuidas de memoria virtual
Distributed World Wide Web information systems	Sistemas distribuidos de información World Wide Web
Division	División
Division operation	Operación división
DKNF (Domain-key normal form)	Forma normal de dominios y claves (FNDC)
DML (Data-manipulation language)	Lenguaje de manipulación de datos (LMD)
DML precompiler	Precompilador del LMD
Document	Documento
Domain	Dominio
Domain constraints	Restricciones de dominio
Domain relational calculus	Cálculo relacional de dominios
Domain types	Tipos de dominios
Domain variable	Variable de dominio
Domain-key normal form (DKNF)	Forma normal de dominios y claves (FNDC)
Domain-key normal form (DKNF) database	Base de datos en forma normal de dominios y claves (FNDC)
Dominant entity	Entidad dominante
Downsizing	Fraccionamiento de costes
Drill down	Concreción
Drop	Eliminar
Drop indices	Supresión de índices
Dummy	Ficticio
Dummy junction type	Tipo de conexión ficticia
Dummy link type	Tipo de enlace ficticio
Dummy record type	Tipo de registro ficticio
Dump (*n.*)	Volcado
Dump (*v.*)	Volcar
Duplicate elimination	Eliminación de duplicados
Durability	Durabilidad
DVD (Digital video disk)	Videodisco digital (VDD)
Dynamic	Dinámico
Dynamic hashing	Asociación dinámica
Dynamic hashing techniques	Técnicas de asociación dinámica
Eager	Impaciente
ECC (Memory-style error-correcting-code organization)	Organización de códigos de corrección de errores tipo memoria
Edge	Arco
EEPROM (Electrically erasable programmable read-only memory)	Memoria sólo de lectura programable y borrable eléctricamente
Efficient	Eficiente

Election	Elección
Election algorithm	Algoritmo de elección
Electrically erasable programmable read-only memory (EEPROM)	Memoria sólo de lectura programable y borrable eléctricamente
Elevator algorithm	Algoritmo del ascensor
Embedded	Incorporado
Embedded language	Lenguaje incorporado
Embedded query language	Lenguaje de consulta incorporado
Embedded SQL language	Lenguaje SQL incorporado
Encapsulating	Encapsulamiento
Encina monitor	Monitor Encina
Encrypt	Cifrar
Encryption	Cifrado
Encryption key	Clave de cifrado
End-of-record	Fin-de-registro
Enterprise schema	Esquema de la empresa
Entity	Entidad
Entity set	Conjunto de entidades
Entity-relationship attribute inheritance	Herencia de atributos entidad-relación
Entity-relationship data model	Modelo de datos entidad-relación
Entity-relationship database schema	Esquema de bases de datos entidad-relación
Entity-relationship diagram	Diagrama entidad-relación
Entity-relationship generalization	Generalización entidad-relación
Entity-relationship model	Modelo entidad-relación
Equality	Igualdad
Equality on key	Igualdad basada en la clave
Equality on nonkey	Igualdad basada en un atributo no clave
Equality-generating dependency	Dependencia de generación de igualdad
Equi-join	Equirreunión
Equivalence rules	Reglas de equivalencia
Equivalent views	Vistas equivalentes
E-R diagram	Diagrama E-R
E-R model (Entity-relationship model)	Modelo E-R (entidad-relación)
Escalable system	Sistema ampliable
Estimate	Estimar
Ethernet	Ethernet
Evaluation algorithms queries	Algoritmos de evaluación de consultas
Evaluation primitives	Primitivas de evaluación
Event	Evento/Suceso
Event-condition-action model	Modelo suceso-condición-acción
Example element	Elemento ejemplo
Example rows	Filas ejemplo

Except operation	Operación except
Exclusive lock	Bloqueo exclusivo
Exclusive mode lock	Bloqueo en modo exclusivo
Exchange-operator model	Modelo de operador de intercambio
Execution of workflows	Ejecución de los flujos de trabajo
Execution skew	Sesgo de ejecución
Execution states	Estados de ejecución
Existence dependencies	Dependencias de existencia
Existence dependency	Dependencia de existencia
Explicit	Explícito
Extendable hashing	Asociación extensible
Extension	Extensión
External	Externo
External merge-sort algorithm	Algoritmo de mezcla-ordenación externa
External sorting	Ordenación externa
External sort-merge	Mezcla-ordenación externa
External variable	Variable externa
Extraneous	Raro
Fact	Hecho
Fail-stop assumption	Supuesto de fallo-parada
Failure detection	Detección de fallos
Failure recovery	Recuperación de fallos
Failure-atomicity	Atomicidad ante fallos
False cycles	Ciclos falsos
False drop	Omisión incorrecta
False positive	Inclusión innecesaria
Fan-out	Grado de salida
Fault tolerance	Tolerancia ante fallos
File	Archivo
File grid	Archivo en retícula
File header	Cabecera de archivo
File organization	Organización de archivos
File organization techniques	Técnicas de organización de archivos
File scan	Explorador de archivo
File structure	Estructura de archivos
File system	Sistema de archivos
File transfer protocol (FTP)	Protocolo de transferencia de archivos
File-processing system	Sistema de procesamiento de archivos
Fine granularity	Granularidad fina
Fine-grain machine	Máquina de grano fino
Fine-granularity parallel machines	Máquinas paralelas de grano fino
First normal form (1NF)	Primera forma normal (1FN)

Fixed-head disk	Disco de cabezas fijas
Fixed-length records	Registros de longitud fija
Fixed-length representation	Representación en longitud fija
Fixpoint	Punto fijo
Flash memory	Memoria *flash*
Floppy disks	Disquetes
Forced output	Salida forzada
Foreign key	Clave externa
Forest protocol	Protocolo de bosque
Form	Formulario
Form interface	Interfaz para formularios
Formal standard	Norma formal
Fortran	Fortran
Forwarding address	Dirección de entrega
Fourth normal form (4NF)	Cuarta forma normal (4FN)
Fourth-generation language (4GL)	Lenguaje de cuarta generación (L4G)
Fragment (*n.*)	Fragmento
Fragment (*v.*)	Fragmentar
Fragment and replicate	Fragmentos y réplicas
Fragment-and-replicate join	Reunión con fragmentos y réplicas
Fragmentation	Fragmentación
Frame-memory model	Modelo de memoria por marcos
Free list	Lista libre
From operation	Operación from
Front-end	Parte visible al usuario
FTP (File transfer protocol)	Protocolo de transferencia de archivos
Fudge factor	Factor de escape
Full outer join	Reunión externa completa
Full replication	Réplica completa
Full-text index	Índice del texto completo
Fully distributed deadlock-detection algorithm	Algoritmo de detección de interbloqueos completamente distribuido
Function hashing	Asociación de funciones
Functional dependencies	Dependencias funcionales
Functional dependency	Dependencia funcional
Fuzzy	Difuso
Fuzzy checkpoint	Punto de revisión difuso
Fuzzy dump	Volcado difuso
Fuzzy dump schemes	Esquemas de volcado difuso
Gamma system	Sistema Gamma
Garbage	Basura
Garbage collection	Recogida de basura

General constraint	Restricción general
Generalized-projection	Proyección generalizada
Generalized-projection operation	Operación proyección generalizada
Geographic data	Datos geográficos
Geographic databases	Bases de datos geográficas
Geographic information systems	Sistemas de información geográfica
Geometric information	Información geométrica
Global positioning system (GPS)	Sistema de localización global
Global transaction	Transacción global
Global-read protocol	Protocolo de lectura global
Global-read-write protocol	Protocolo global de lectura y escritura
Global-read-write/local-read protocol	Protocolo de lectura y escritura global y lectura local
Gopher	Gopher
GPS (Global positioning system)	Sistema de localización global
Grace system	Sistema Grace
Grant	Concesión
Grant operation	Operación grant
Granting of locks	Concesión de bloqueos
Granting of privileges	Concesión de privilegios
Granularity	Granularidad
Granularity locks	Granularidad de los bloqueos
Graph	Grafo
Graph precedence	Precedencia de grafos
Graph-based protocol	Protocolo basado en grafos
Graphical data	Datos gráficos
Graphical user interface	Interfaz gráfica de usuario
Grid array	*Array* en retícula
Grid file	Archivo en retícula
Ground instantation	Ejemplar básico
Ground instantiation of a rule	Ejemplar básico de una regla
Group	Grupo
Group by operation	Operación group by
Group-commit technique	Técnica de compromiso en grupo
Grouping	Agrupación
Growing phase	Fase de crecimiento
Gupta SQL languages	Lenguaje SQL Gupta
Handoff	Relevo
Handoff of control	Relevo del control
Hard disk	Disco rígido
Hardware swizzling	Rescate hardware
Harmonic mean	Media armónica
Hash	Asociar

Hash function	Función de asociación
Hash index	Índice asociativo
Hash partitioning	División por asociación
Hashing	Asociación
Hashing file organization	Organización asociativa de archivos
Hash-table overflow	Desbordamiento de una tabla de asociación
Head	Cabeza
Head-disk assemblies	Dispositivos cabeza-disco
Header	Cabecera
Heap file organization	Organización de archivos en montículo
Heterogeneous distributed database systems	Sistemas distribuidos y heterogéneos de bases de datos
Heterogeneous distributed databases	Bases de datos distribuidas y heterogéneas
Heuristic	Heurística
Hidden pointers	Puntero oculto
Hierarchical architecture	Arquitectura jerárquica
Hierarchical databases	Bases de datos jerárquicas
Hierarchical model	Modelo jerárquico
Hierarchy	Jerarquía
High availability	Disponibilidad elevada
Higher-level	Nivel más alto
Higher-level entity set	Conjunto de entidades de nivel superior
High-performance transaction systems	Sistemas de transacciones de alto rendimiento
High-performance transactions	Transacción de alto rendimiento
Histogram	Histograma
Histogram graph	Grafo de histograma
Holds on *R*	Se cumple en *R*
Home page	Página inicial
Homogeneous distributed database	Base de datos distribuida homogénea
Horizontal fragmentation	Fragmentación horizontal
Horizontal partitioning	División horizontal
Host	Anfitrión
Host computer	Computadora anfitriona
Host language	Lenguaje anfitrión
Hot-spare configuration	Configuración de relevo en caliente
HTML (Hypertext markup language)	Lenguaje de marcas de hipertexto
HTTP (HyperText transfer protocol)	Protocolo para transferencia de hipertexto
Hybrid hash-join algorithm	Algoritmo de reunión por asociación híbrida
Hybrid merge-join algorithm	Algoritmo híbrido de reunión por mezcla
HyperCard	HyperCard
Hypercube interconnection	Interconexión hipercubo
Hypermedia systems	Sistemas hipermedia
Hypertext	Hipertexto

Hypertext databases	Bases de datos de hipertexto
Hypertext markup language (HTML)	Lenguaje de marcas de hipertexto
HyperText transfer protocol (HTTP)	Protocolo para transferencia de hipertexto
I/O parallelism	Paralelismo de E/S
IBM Scientific Center	Centro científico de IBM
IBM T.J. Watson Research Center	Centro de investigación T. J. Watson de IBM
Idempotent	Idempotente
Identify	Identificar
Identifying relationship	Relación de identificación
IDL (Interface description language)	Lenguaje de descripción de interfaces
IEEE (Institute for electrical and electronical engineers)	Instituto de ingenieros eléctricos y electrónicos
Illustra database system	Sistema de bases de datos Illustra
Immediate	Inmediata
Immediate-modifications scheme	Esquema de modificación inmediata
Immediate-update technique	Técnica de actualización inmediata
Implement	Implementar
Implementation	Implementación
Implicit	Implícito
IMS (Information management system)	Sistema gestor de información de IBM
IMS Fast Path	Fast Path de IMS
IMS monitor	Monitor IMS
Inconsistent state	Estado inconsistente
Independent	Independiente
Independent parallelism	Paralelismo independiente
Index	Índice
Index authorization	Autorización de índice
Index B^+-tree	Índice del árbol B^+
Index entry	Entrada del índice
Index hashing	Asociación de índice
Index locking	Bloqueo del índice
Index record	Registro del índice
Index scans	Exploraciones del índice
Indexed nested-loop join	Reunión en bucle anidado indexada
Indexing documents	Creación de índices de documentos
Indexing spatial data	Creación de índices de datos espaciales
Index-locking technique	Técnica de bloqueo de índice
Index-sequential file	Archivo secuencial indexado
In-doubt transaction	Transacción dudosa
Inexpensive	Bajo coste
Infer	Inferir
Information management system (IMS)	Sistema gestor de información de IBM
Information retrieval	Recuperación de la información

Ingres database system	Sistema de bases de datos Ingres
Inheritance	Herencia
In-memory pointers	Punteros internos de memoria
In-memory tree structure	Estructura de árbol en memoria
Inner (join)	Interna (reunión)
Insertion	Inserción
Instance	Ejemplar
Instance variable	Variable de ejemplares
Instantiation	Ejemplar
Institute for Electrical and Electronical Engineers (IEEE)	Instituto de Ingenieros Eléctricos y Electrónicos
Integrated services digital network (ISDN)	Red digital de servicios integrados (RDSI)
Integrity	Integridad
Integrity constraints	Restricciones de integridad
Intention lock	Bloqueo intencional
Intention lock mode	Modo de bloqueo intencional
Intention-exclusive (IX) mode	Modo intencional-exclusivo (IX)
Intention-shared (IS) mode	Modo intencional-compartido (IC)
Interconnection network	Red de interconexión
Interface	Interfaz (*f*.)
Interface description language (IDL)	Lenguaje de descripción de interfaces
Interference	Interferencia
Internal	Interno
International standards organization (ISO)	Organización internacional de normalización
Internet	Internet
Interoperation parallelism	Paralelismo entre operaciones
Interquery parallelism	Paralelismo entre consultas
Intersect operation	Operación intersect
Intersection	Intersección
Intersection operation	Operación intersección
Intersection rule	Regla de la intersección
Interval	Intervalo
Intraoperation parallelism	Paralelismo en operaciones
Intraquery parallelism	Paralelismo en consultas
Invalidation report	Informe de invalidación
Inverted index	Índice invertido
Invoke a method	Invocar un método
ISDN (Integrated services digital network)	Red digital de servicios integrados (RDSI)
ISO (International Standards Organization)	Organización Internacional de Normalización
Isochronous data	Datos isócronos
Isolation	Aislamiento
Iterator	Iterador
Java	Java

JCL (Job control language)	Lenguaje de control de trabajos
Job control language (JCL)	Lenguaje de control de trabajos
Join	Reunión
Join dependencies	Dependencias de reunión
Join operation	Operación join
Join partitioning	Reunión fraccionada
Joint picture experts group (JPEG)	Grupo conjunto de expertos en imágenes
JPEG (Joint picture experts group)	Grupo conjunto de expertos en imágenes
Jukebox	Cambiador automático
k-d tree	Árbol k-d
k-d-B tree	Árbol k-d-B
Key	Clave
Keyword	Palabra clave
Keyword-based information retrieval	Recuperación de información basada en palabras clave
Kill	Cancelar
Knowledge-discovery systems	Sistemas de descubrimiento del conocimiento
Labeled graph precedence	Precedencia de grafos etiquetados
Labeled precedence graph	Grafo de precedencia etiquetado
LAN (Local-area network)	Red de área local
Large data items recoverability	Recuperabilidad de elementos de datos de gran tamaño
Latch	Pestillo
Latency time	Tiempo de latencia
Lattice	Retículo
Layer	Capa
Least recently used (LRU)	Utilizado menos recientemente
Left outer join	Reunión externa por la izquierda
Left-deep join orders	Órdenes de reunión en profundidad por la izquierda
Leftmost-child tree	Árbol hijo más a la izquierda
Legacy systems	Sistemas heredados
Legal	Legal
Lexicographic ordering	Orden lexicográfico
Line	Línea
Linear hashing	Asociación lineal
Linear probing	Ensayo lineal
Linear scales	Escalas lineales
Linear scaleup	Ampliabilidad lineal
Linear search	Búsqueda lineal
Linear speedup	Ganancia de velocidad lineal
Link	Enlace
Link failure	Fallo de enlace
Linked list	Lista enlazada
Little-endian form	Orden menos significativo

Load-balanced partition vector	Vector de división con carga equilibrada
Load-balanced partitioning	División con carga equilibrada
Local autonomy	Autonomía local
Local transaction	Transacción local
Local transaction manager	Gestor de transacciones locales
Local wait-for graph	Grafo de espera local
Local-area network (LAN)	Red de área local
Local-read protocol	Protocolo de lectura local
Lock	Bloqueo
Lock conversions	Conversiones de bloqueo
Lock deescalation	Liberación de bloqueos
Lock point	Punto de bloqueo
Lock protocol	Protocolo basado en el bloqueo
Locking protocol	Protocolo de bloqueo
Log	Registro histórico
Log disk	Disco de registro histórico
Log force	Forzar el registro histórico
Log-based failure recovery	Recuperación de fallos basada en el registro histórico
Log-based file system	Sistema de archivos basado en el registro histórico
Logical clock	Reloj lógico
Logical connectives	Conectivas lógicas
Logical counter	Contador lógico
Logical error	Error lógico
Logical level	Nivel lógico
Logical logging	Registro histórico lógico
Logical object identifier	Identificador lógico de objeto
Logical operation	Operación lógica
Logical schema	Esquema lógico
Logical-design	Diseño lógico
Logically implied	Implicadas lógicamente
Log-record buffering	Registro histórico con memoria intermedia
Long fields	Campos largos
Long-distance computer networks	Redes informáticas de larga distancia
Long-duration transaction	Transacción de larga duración
Lookup operation	Operación lookup
Lossless join	Reunión sin pérdida
Lossless-join decomposition	Descomposición de reunión sin pérdida
Lossy decomposition	Descomposición con pérdida
Lossy join	Reunión con pérdida
Lossy-join decomposition	Descomposición de reunión con pérdida
Lotus Notes	Lotus Notes
Lower-level	Nivel más bajo

LRU (Least recently used)	Utilizado menos recientemente
LRU block-replacement scheme	Esquema de sustitución de bloques LRU
Magnetic tapes	Cintas magnéticas
Magnetic-disk storage	Almacenamiento en discos magnéticos
Main memory	Memoria principal
Main-memory database system	Sistema de bases de datos en memoria principal
Many to many mapping	Correspondencia de varios a varios
Many to one mapping	Correspondencia de varios a uno
Many-server, many-router model	Modelo de varios servidores y varios encaminadores
Many-server, single-router model	Modelo de varios servidores y un solo encaminador
Many-server, single-router routing	Encaminamiento con varios servidores y un solo servidor
Mapping	Correspondencia
Mapping cardinalities	Correspondencia de cardinalidades
Mart	Puesto
Massively parallel machine	Máquina masivamente paralela
Massively parallel system	Sistema masivamente paralelo
Materialization	Materialización
Materialize	Materializar
Materialized evaluation	Evaluación materializada
Materialized view	Vista materializada
Mean time to data loss	Tiempo medio entre pérdidas de datos
Mean time to failure	Tiempo medio entre fallos
Mean time to repair	Tiempo medio de reparación
Member (or child)	Miembro (o hijo)
Memory-style error-correcting-code organization (ECC)	Organización de códigos de corrección de errores tipo memoria
Merge	Mezclar
Merge-join algorithm	Algoritmo de reunión por mezcla
Mesh	Trama
Message	Mensaje
Metadata	Metadatos
Method	Método
Microsoft Access databases	Bases de datos de Microsoft Access
Mini-batch transaction	Transacción por minilotes
Minimum cost	Coste mínimo
Mirror disk	Disco imagen
Mirroring	Creación de imágenes
Missing	Perdido
Mixed fragmentation	Fragmentación mixta
Mobile computer	Computadora portátil
Mobile databases	Bases de datos portátiles
Mobile support stations	Estaciones de apoyo para computadoras portátiles
Mobile-computing	Informática portátil

Modem	Módem
Modification of databases	Modificación de bases de datos
Modular-stratification semantics	Semántica de estratificación modular
Module	Módulo
Monitor	Observar
Monotonic	Monótona
Monotonic function	Función monótona
Monotonic systems	Sistemas monótonos
Most recently used (MRU)	Utilizado más recientemente
Most specific type	Tipo más específico
Motion picture experts group (MPEG)	Grupo de expertos en películas
MPEG (Motion picture experts group)	Grupo de expertos en películas
MPEG-1 standard	Norma MPEG-1
MPEG-2 standard	Norma MPEG-2
MRU (Most recently used)	Utilizado más recientemente
Multidatabase system	Sistema con múltiples bases de datos
Multidimensional data	Datos multidimensionales
Multilevel	Multinivel
Multilevel index	Índice multinivel
Multilevel transaction	Transacción multinivel
Multimedia data formats	Formatos de datos multimedia
Multimedia databases	Bases de datos multimedia
Multiple application-server processes	Procesos del servidor para varias aplicaciones
Multiple granularity	Granularidad múltiple
Multiple inheritance	Herencia múltiple
Multiple transactions	Varias transacciones
Multiple-coordinator approach	Enfoque de varios coordinadores
Multiple-granularity locking protocol	Protocolo de bloqueo de granularidad múltiple
Multiple-key access	Acceso multiclave
Multiprogramming	Multiprogramación
Multiset	Multiconjunto
Multiset versions	Versiones multiconjunto
Multisystem applications	Aplicaciones multisistema
Multitasking	Multitarea
Multithreading	Multienhebramiento
Multiuser systems	Sistema multiusuario
Multivalued	Multivalorado
Multivalued attributes	Atributos multivalorados
Multivalued augmentation rule	Regla de la aumentatividad multivalorada
Multivalued dependency	Dependencia multivalorada
Multivalued transitivity rule	Regla de la transitividad multivalorada
Multivalued union rule	Regla de la unión multivalorada

Multiversion	Multiversión
Multiversion concurrency-control scheme	Esquema de control de concurrencia multiversión
Multiversion database systems	Sistemas de bases de datos multiversión
Multiversion timestamp-ordering scheme	Esquema de ordenación por marcas temporales multiversión
Multiversion two-phase locking protocol	Protocolo de bloqueo de dos fases multiversión
Naive user	Usuario normal
Name server	Servidor de nombres
Natural join	Reunión natural
Navigational model	Modelo de navegación
Nearest-neighbor query	Consulta de vecino más próximo
Nearness queries	Consultas de proximidad
Negative literal	Literal negativo
Nested relational	Relacional anidado
Nested relational model	Modelo relacional anidado
Nested transaction	Transacción anidada
Nested transactions	Transacciones anidadas
Nested-loop join	Reunión en bucle anidado
Nesting	Anidamiento
Network database	Base de datos en red
Network model	Modelo de red
Network partitioning	División de la red
Network transparency	Transparencia de la red
Network-level security	Seguridad en el nivel de la red
Nonacceptable termination state	Estado de terminación no aceptable
Nonatomic types	Tipos no atómicos
Nonclustering index	Índice sin agrupación
Nonprocedural language	Lenguaje no procedimental
Nonrecoverable schedule	Planificación no recuperable
Nonserializable executions	Ejecuciones no secuenciables
Non-stop system	Sistema ininterrumpido
Nonvolatile random-access memory (nonvolatile RAM)	Memoria no volátil de acceso aleatorio (RAM no volátil)
Nonvolatile storage	Almacenamiento no volátil
Normal form	Forma normal
Normalization	Normalización
Not known	No conocido
NP-complete problems	Problemas NP-completos
Null	Nulo
Null attributes	Atributos nulos
Null values	Valores nulos
Number of block transfers from disk	Número de transferencias de bloques de disco
N-way merge	Mezcla de *n* vías
Object	Objeto

Object classes	Clases de objetos
Object containment	Continente de objetos
Object database management group (ODMG)	Grupo de gestión de bases de datos de objetos
Object definition language (ODL)	Lenguaje de definición de objetos
Object identifier (OID)	Identificador de objeto
Object management architecture (OMA)	Arquitectura para gestión de objetos
Object management group (OMG)	Grupo de gestión de objetos
Object oriented programming (OOP)	Programación orientada a objetos
Object relational databases	Bases de datos relacionales orientadas a objetos
Object relational models	Modelos relacionales orientados a objetos
Object relational systems	Sistemas relacionales orientados a objetos
Object request broquer (ORB)	Agente para peticiones de objetos
Object SQL (OSQL)	Object SQL
Object-based data models	Modelos lógicos basados en objetos
Object-oriented data model	Modelo de datos orientado a objetos
Object-oriented databases	Bases de datos orientadas a objetos
Object-oriented model	Modelo orientado a objetos
Object-relational data models	Modelos de datos relacionales orientados a objetos
Object-relational system	Sistema relacional orientados a objetos
Objects to files mapping	Correspondencia de objetos a archivos
Observable external writes	Escrituras externas observables
ODBC (Open database connectivity)	Conectividad abierta de bases de datos
ODL (Object definition language)	Lenguaje de definición de objetos
Office information systems (OIS)	Sistemas de información para oficinas
Off-line	Sin conexión
Off-line storage	Almacenamiento sin conexión
Offset	Desplazamiento
OID (Object identifier)	Identificador de objeto
OIS (Office information systems)	Sistemas de información para oficinas
OLAP (On-line analytical processing)	Procesamiento en conexión analítico
OLTP (On-line transaction processing)	Procesamiento en conexión de transacciones
OMA (Objects management architecture)	Arquitectura para gestión de objetos
OMG (Objects management group)	Grupo de gestión de objetos
One to many mapping	Correspondencia de uno a varios
One to one mapping	Correspondencia de uno a uno
One-safe	Uno seguro
On-line	Con conexión/Interactivo
On-line analytical processing (OLAP)	Procesamiento en conexión analítico
On-line encyclopedias	Enciclopedias interactivas
On-line storage	Almacenamiento en conexión
On-line transaction processing (OLTP)	Procesamiento en conexión de transacciones
OOP (Object oriented programming)	Programación orientada a objetos

Open database connectivity (ODBC)	Conectividad abierta de bases de datos
Open hashing	Asociación abierta
Operator tree	Árbol de operadores
Optical disks	Discos ópticos
Optical storage	Almacenamiento óptico
Or	Disyunción
Oracle 7	Oracle 7
ORB (Object request broquer)	Agente para peticiones de objetos
Ordered index	Índice ordenado
Organize documents logically	Organizar lógicamente los documentos
OSQL (Object SQL)	Object SQL
Outer	Externa
Outer-join	Reunión externa
Output values	Valores de salida
Overflow block	Bloque de desbordamiento
Overflow buckets	Cajones de desbordamiento
Overflow chaining	Cadena de desbordamiento
Overlaps	Solapa
Overloading	Sobrecarga
Own	Propietario
Owner (or parent)	Propietario (o padre)
P+Q redundancy scheme	Esquema de redundancia P+Q
Page	Página
Page fault	Fallo de página
Page table	Tabla de páginas
Parallel architecture	Arquitectura paralela
Parallel database architectures	Arquitecturas paralelas de bases de datos
Parallel database system	Sistema paralelo de bases de datos
Parallel external sort-merge	Mezcla-ordenación paralela externa
Parallel join	Reunión paralela
Parallel join algorithm	Algoritmo de reunión paralela
Parallel sort	Ordenación paralela
Parallel system	Sistema paralelos
Parameter	Parámetro
Parent (or owner)	Padre (o propietario)
Partial	Parcial
Partial key	Clave parcial
Partially connected network	Red conectada parcialmente
Partially dependent	Dependiente parcialmente
Participation	Participación
Partition (*n.*)	Partición
Partition (*v.*)	Dividir

Partition skew	Sesgo de la división
Partitioned hashing	Asociación dividida
Partitioned parallelism	Paralelismo de particiones
Pascal	Pascal
Pass	Ciclo
Password	Contraseña
Pathway	Pathway
Performance	Rendimiento
Performance benchmark	Prueba de rendimiento
Performance tuning	Ajuste del rendimiento
Performance-simulation model	Modelo de simulación de rendimiento
Persistence of objects	Persistencia de los objetos
Persistent extensions	Extensiones persistentes
Persistent pointer	Puntero persistente
Persistent programming language	Lenguaje de programación persistente
Phantom phenomenon	Fenómeno fantasma
Physical	Físico
Physical block	Bloque físico
Physical data model	Modelo de datos físicos
Physical level	Nivel físico
Physical log record	Registro físico del registro histórico
Physical logging	Registro histórico físico
Physical object identifier	Identificador físico de objeto
Physical schema	Esquema físico
Physical-design	Diseño físico
Pinned	Clavado
Pipeline	Cauce
Pipelined join	Reunión encauzada
Pipelined parallelism	Paralelismo de encauzamiento
Pipelining	Encauzamiento
PJNF (Project-join normal form)	Forma normal de reunión por proyección (FNRP)
Platter	Plato
Point	Apuntar
Point queries	Consultas concretas
Pointer	Puntero
Pointer fields	Campo puntero
Pointer swizzling	Rescate de punteros
Polling	Encuestas
Population	Población
Positive literal	Literal positivo
Possibility	Posibilidad
Postgres	Postgres

PostScript	PostScript
PowerBuilder	PowerBuilder
PR quadtree	Árbol cuadrático PR
Precede	Preceder
Precedence graph	Grafo de precedencia
Precision	Precisión
Preempt	Expropiar
Prefetching	Preextracción
Presentation facilities	Utilidades para presentaciones
Preserve equivalence	Preservar la equivalencia
Presume abort	Suponer aborto
Presume commit	Suponer compromiso
Price per TPS	Precio por TPS
Primary copy	Copia principal
Primary index	Índice primario
Primary key	Clave primaria
Primary site	Emplazamiento principal
Primary storage	Almacenamiento principal
Prime	Primo
Private key	Clave privada
Privilege	Privilegio
Privilege list	Lista de privilegios
Probe	Probar
Probe input	Entrada para probar
Procedural language	Lenguaje procedimental
Process	Proceso
Processing	Procesamiento
Processing entity	Entidad de procesamiento
Process-per-client model	Modelo de proceso por cliente
Producer-driven pipelining	Encauzamiento por los productores
Program	Programa
Project	Proyectar
Project join	Reunión por proyección
Project operation	Operación proyección
Projection	Proyección
Project-join normal form (PJNF)	Forma normal de reunión por proyección (FNRP)
Prolog	Prolog
Pseudotransitivity rule	Regla de la pseudotransitividad
Public key	Clave pública
Public-key encryption	Cifrado de clave pública
Pulling	Extracción
Pushing	Inserción

QBE (Query-by-example)	Consulta mediante ejemplos
QMF (Queries management facility)	Recurso de gestión de consultas
Quadtree	Árbol cuadrático
Quasi-serializability	Cuasisecuencialidad
Quel	Quel
Queries relations	Relaciones de consultas
Queries server systems	Sistemas servidores de consultas
Query	Consulta
Query language	Lenguaje de consultas
Query management facility (QMF)	Recurso de gestión de consultas
Query optimization	Optimización de consultas
Query processing	Procesamiento de consultas
Query processor	Procesador de consultas
Query range	Rango de las consultas
Query-by-example (QBE)	Consulta mediante ejemplos
Query-evaluation algorithm	Algoritmo para la evaluación de consultas
Query-evaluation plan	Plan de evaluación de la consulta
Query-execution engine	Motor de ejecución de consultas
Query-execution plan	Plan de ejecución de la consulta
Query-server system	Sistema servidor de consultas
Queue	Cola
Queue manager	Gestor de colas
Queue system	Sistema de colas
Queue theory	Teoría de colas
Queueing systems	Sistemas de colas
Quicksort	Ordenación rápida
RAID (Redundant array of independent disks)	Disposición redundante de discos independientes
RAID levels	Niveles de RAID
Range of operation	Operación range of
Range partitioning	División por rangos
Range query	Consulta de rangos
Raster	Por líneas
Reactionary standard	Norma reaccionaria
Read authorization	Autorización de lectura
Read commited	Con compromiso de lectura
Read phase	Fase de lectura
Read uncommited	Sin compromiso de lectura
Read-only	Sólo de lectura
Read-write head	Cabeza de lectura y escritura
Real graph	Grafo real
Real-time system	Sistema de tiempo real
Rebuild performance	Rendimiento de reconstrucción

Recall	Recuperación
Reconfigure	Volver a configurar
Record	Registro
Record type Rlink	Tipo de registro Renlace
Record-based data models	Modelos lógicos basados en registros
Recover	Recuperar
Recoverability	Recuperabilidad
Recoverability schedule	Recuperabilidad de planificaciones
Recoverable schedules	Planificaciones recuperables
Recovery	Recuperación
Recovery algorithm	Algoritmo de recuperación
Recovery scheme	Esquema de recuperación
Recovery-management component	Componente para la gestión de la recuperación
Recursion	Recursividad
Recursive	Recursivo
Recursive partitioning	División recursiva
Redistribute	Redistribuir
Redo	Rehacer
Redo log record	Registro rehacer del registro histórico
Redo operation	Operación redo
Redo phase	Fase rehacer
Redo-list	Lista-rehacer
Redundancy	Redundancia
Redundant array of independent disks (RAID)	Disposición redundante de discos independientes
Redundant array of inexpensive disks systems	Sistemas RAID
Reed-Solomon codes	Códigos Reed-Solomon
Reference	Referencia
Reference privilege	Privilegio de referencias
Reference types	Tipos de referencia
Referencial-integrity constraints	Restricciones de integridad referencial
Referential integrity	Integridad referencial
Referential integrity constraint	Restricción de integridad referencial
Reflexivity rule	Regla de la reflexividad
Region quadtrees	Árboles cuadráticos con regiones
Region queries	Consultas regionales
Relation	Relación
Relation instance	Ejemplar de relación
Relation schema	Esquema de la relación
Relational algebra	Álgebra relacional
Relational data models	Modelos de datos relacionales
Relational databases	Bases de datos relacionales
Relational model	Modelo relacional

Relational operation	Operación relacional
Relational system	Sistema relacional
Relational-algebra expression	Expresión del álgebra relacional
Relationship	Relación
Relationship set	Conjunto de relaciones
Reliability	Fiabilidad
Remapping of bad sectors	Reasignación de los sectores dañados
Remote backup site	Nodo remoto copia de seguridad
Remote backup systems	Sistemas remotos de copia de seguridad
Remote-procedure-call (RPC)	Llamada a procedimientos remotos
Remote-procedure-call mechanism	Mecanismo de llamadas a procedimientos remotos
Rename	Renombrar
Rename operation	Operación rename
Renaming	Renombramiento
Reorganize	Reorganizar
Repeatable read	Lectura repetible
Repeating history	Repetición de la historia
Repetition of information	Repetición de información
Replace operation	Operación replace
Replication rule	Regla de réplicas
Report writers	Diseñadores de informes
Request	Solicitud
Requirement	Requisito
Resolution	Resolución
Resource authorization	Autorización de recursos
Resource manager	Gestor de recursos
Response	Respuesta
Restart recovery	Recuperación al reiniciar
Restriction	Restricción
Revoke statement	Instrucción revoke
Right outer join	Reunión externa por la derecha
Rigorous two-phase locking protocol	Protocolo de bloqueo riguroso de dos fases
Ring structure	Estructura de anillo
Robustness	Robustez
Role	Papel
Rollback (*n.*)	Retroceso
Rollback (*v.*)	Retroceder
Rollup	Abstracción
Rooted tree	Árbol con raíz
Rotational latency time	Tiempo de latencia rotacional
Round-robin	Turno rotatorio
Round-robin partitioning	División por turno rotatorio

Route	Encaminar
Router	Encaminador
Routing	Encaminamiento
Row	Fila
RPC (Remote-procedure-call)	Llamada a procedimientos remotos
RTR monitors	Monitores RTR
R-tree	Árbol R
Rule	Regla
Run	Secuencia
SAA-SQL (System application architecture database interface)	Interfaz de bases de datos para arquitecturas de aplicación a sistemas
Safety	Seguridad
Saga	Saga
San José Research Center	Laboratorio de Investigación de San José
Satisfy	Satisfacer
Scaleup	Ampliabilidad
Scanning a relation	Exploración de una relación
SCSI (Small computer-system interconnect)	Interconexión de pequeños sistemas informáticos
Schedule	Planificar
Scheduling	Planificación
Schema	Esquema
Search key	Clave de búsqueda
Second normal form (2NF)	Segunda forma normal (2FN)
Secondary index	Índice secundario
Secondary site	Nodo secundario
Secondary storage	Almacenamiento secundario
Sector	Sector
Security specification	Especificación de seguridad
Seek time	Tiempo de búsqueda
Segmentation violation	Violación de la segmentación
Select	Seleccionar
Select operation	Operación select
Selection	Selección
Selectivity	Selectividad
Semantics of a program	Semántica de un programa
Semantics of a rule	Semántica de una regla
Semi join	Semirreunión
Semijoin strategy	Estrategia de semirreunión
Semistructured data	Datos semiestructurados
Sending a message	Paso de mensaje
Sentence	Instrucción
Sequential file	Archivo secuencial

Sequential file organization index	Índice de organización de archivos secuenciales
Sequential scan	Búsqueda secuencial
Sequential-access storage	Almacenamiento de acceso secuencial
Serial	Secuencial
Serializability	Secuencialidad
Serializability order	Orden de secuencialidad
Server system	Sistema servidor
Service time	Tiempo de servicio
Set comparison	Comparación de conjuntos
Set difference	Diferencia de conjuntos
Set intersection	Intersección de conjuntos
Set membership	Pertenencia al conjunto
Set occurrence	Aparición del conjunto
Set operations	Operaciones de conjuntos
Set-difference operation	Operación diferencia de conjuntos
Set-top box computer	Microcomputadora
SGML (Standard generalized markup language)	Lenguaje normalizado de marcas generalizado
Shadow copy	Copia en la sombra
Shadow page table	Tabla de páginas sombra
Shadow paging	Paginación en la sombra
Shadowing	Creación de sombras
Shared and intention-exclusive (SIX) mode	Modo intencional-exclusivo y compartido (IXC)
Shared lock	Bloqueo compartido
Shared memory	Memoria compartida
Shared mode locks	Modo de bloqueo compartido
Shared-disk architecture	Arquitectura de disco compartido
Shared-disk model	Modelo de disco compartido
Shared-disk system	Sistema de disco compartido
Shared-memory architecture	Arquitectura de memoria compartida
Shared-memory multiprocessor system	Sistema multiprocesador de memoria compartida
Shared-nothing architecture	Arquitectura sin compartimiento
Short-duration transaction	Transacción de corta duración
Shrinking phase	Fase de decrecimiento
Similar	Semejante
Similarity-based retrieval	Recuperación basada en la semejanza
Simula	Simula
Single relation schema	Esquema de rotación única
Single user system	Sistema de usuario único
Single valued	Univalorados
Single-server model	Modelo de servidor único
Site	Nodo/Emplazamiento
Skeleton tables	Esqueletos de tablas

Skew (*n.*)	Sesgo
Skew (*v.*)	Sesgar
Skew partitioning	Sesgo de la división
Slotted-page structure	Estructura de páginas con ranuras
Small computer-system interconnect (SCSI)	Interconexión de pequeños sistemas informáticos
Smalltalk	Smalltalk
Snapshot relation	Relación instantánea
Software swizzling	Rescate software
Sort-merge join	Reunión por mezcla-ordenación
Sort-merge-join algorithm	Algoritmo de reunión por mezcla-ordenación
Sound	Correcto
Sound rules	Reglas correctas
Span	Duración
Sparse index	Índice disperso
Spatial databases	Bases de datos espaciales
Spatial join	Reunión espacial
Spatial queries	Consultas espaciales
Specialization	Especialización
Specification	Especificación
Specification of functional requirements	Especificación de requisitos funcionales
Specification of user requirements	Especificación de requisitos del usuario
Speedup	Ganancia de velocidad
Split	Dividir
SQL (Structured query language)	Lenguaje estructurado de consultas
SQL Access Group	Grupo de acceso SQL
SQL language reference privilege	Privilegios de referencia del lenguaje SQL
SQL language referencial integrity	Integridad referencial en el lenguaje SQL
SQL language standard	Norma del lenguaje SQL
Stable storage	Almacenamiento estable
Standard	Norma
Standard generalized markup language (SGML)	Lenguaje normalizado de marcas generalizado
Start	Iniciar
Startup	Inicio
Startup time	Tiempo de inicio
Starvation	Inanición
State of a workflow	Estado de un flujo de trabajo
Static hashing	Asociación estática
Statistical analyses	Análisis estadísticos
Statistical databases	Bases de datos estadísticas
Step	Paso
Stop words	Palabras de parada
Storage manager	Gestor de almacenamiento

Stored	Almacenado
Strict two-phase locking protocol	Protocolo de bloqueo estricto de dos fases
String operations	Operaciones de cadena
Strong correctness	Corrección fuerte
Strong entity set	Conjunto de entidades fuerte
Strong types	Tipos estrictos
Structured query language (SQL)	Lenguaje estructurado de consultas
Subclass	Subclase
Sublinear scaleup	Ampliabilidad sublineal
Sublinear speedup	Ganancia de velocidad sublineal
Submit	Remitir
Subordinate entity	Entidad subordinada
Subquery	Subconsulta
Subschema	Subesquema
Subset dependency	Dependencia de subconjunto
Substitutability	Posibilidad de sustitución
Successful completion	Terminación con éxito
Sufficient conditions	Condiciones suficientes
Summary	Resumen
Superclass	Superclase
Superkey	Superclave
Superuser	Superusuario
Support	Soporte
Swap space	Espacio de intercambio
Swizzling	Rescate
Synonym	Sinónimo
Synthesize	Sintetizar
System application architecture database interface (SAA-SQL)	Interfaz de bases de datos para arquitecturas de aplicación a sistemas
System catalog	Catálogo del sistema
System clock	Reloj del sistema
System crash	Caída del sistema
System error	Error del sistema
System failure	Fallo del sistema
System R	System R
Table	Tabla
Tableau	Tableau
Tableau optimization	Optimización con tableaux
Tabular representation	Representación tabular
Tandem	Tándem
Tape jukebox	Cambiador de cintas
Tape storage	Almacenamiento en cinta

Task	Tarea
Task flow	Flujo de tareas
Teleprocessing monitor	Monitor de teleprocesamiento
Template	Plantilla
Temporal database	Base de datos temporal
Temporal join	Reunión temporal
Temporal projection	Proyección temporal
Temporal query language	Lenguaje de consultas temporales
Temporal relation	Relación temporal
Temporal selection	Selección temporal
Teradata database machine	Máquina de bases de datos de Teradata
Terminal	Terminal
Terminated	Terminado
Termination states	Estados de terminación
Tertiary storage	Almacenamiento terciario
Testing for serializability	Prueba de secuencialidad
Text data	Datos de texto
Text-markup language	Lenguaje de marcas de texto
Theft of information	Robo de información
Theta (θ)	Zeta
Third normal form (3NF)	Tercera forma normal (3FN)
Thomas' write rule	Regla de escritura de Thomas
Three-phase commit (3PC) protocol	Protocolo de compromiso de tres fases (C3F)
Throughput	Productividad
Ticket	Billete
Time stamp	Marca temporal
Time to completion	Tiempo para finalizar
Timeout lock	Bloqueo con límite de tiempo
Timeout scheme	Esquema de tiempo límite
Timestamp-based protocol	Protocolo basado en la marca temporal
Timestamp-ordering protocol	Protocolo de ordenación por marcas temporales
Timestamp-ordering scheme	Esquema de ordenación por marcas temporales
TLB (Translation lookaside buffer)	Memoria intermedia con traducción anticipada
To depend on	Depender de
Token ring	Anillo con paso de testigo
Top-down	Descendente
Topological sorting	Ordenación topológica
Toss-immediate strategy	Estrategia de extracción inmediata
Total	Total
TP (Transaction processing)	Procesamiento de transacciones
TPC (Transaction processing performance council)	Consejo de rendimiento de procesamiento de transacciones
TPS (Transactions per second)	Transacciones por segundo

Track	Pista
Training set	Conjunto de entrenamiento
Transaction	Transacción
Transaction control	Control de las transacciones
Transaction coordinator	Coordinador de transacciones
Transaction failure	Fallo en transacción
Transaction management	Gestión de transacciones
Transaction processing (TP)	Procesamiento de transacciones
Transaction processing performance council (TPC)	Consejo de rendimiento de procesamiento de transacciones
Transaction rollback	Retroceso de la transacción
Transaction scaleup	Ampliabilidad de transacciones
Transaction server system	Sistema servidor de transacciones
Transaction time	Tiempo de transacción
Transaction workflow	Flujo de trabajo de transacciones
Transactional remote procedure call	Llamada a procedimientos remotos de transacciones
Transactional RPC interface	Interfaz RPC para transacciones
Transaction-management component	Componente de gestión de transacciones
Transactions per second (TPS)	Transacciones por segundo
Transaction-server systems	Sistemas servidores de transacciones
Transfer	Transferir
Transfer rate	Velocidad de transferencia
Transient	Transitorio
Transitive closure	Cierre transitivo
Transitive dependencies	Dependencias transitivas
Transitivity rule	Regla de la transitividad
Translation lookaside buffer (TLB)	Memoria intermedia con traducción anticipada
Tree protocol	Protocolo de árbol
Tree structure	Estructura de árbol
Tree-locking protocol	Protocolo de bloqueo del árbol
Tree-structure diagrams	Diagramas de estructura de árbol
Triangulation	Teselación por triangulación
Tries	Tries
Trigger (*n.*)	Disparador
Trigger (*v.*)	Disparar
Triggering event	Suceso disparador
Trivial	Trivial
Trivial multivalued dependency	Dependencia multivalorada trivial
Tunable parameters	Parámetros ajustables
Tuning	Ajuste
Tuple	Tupla
Tuple relational calculus	Cálculo relacional de tuplas
Tuple variable	Variable tupla

Tuple-generating dependency	Dependencia de generación de tuplas
Tuple-id	Id-tupla
Two-dimensional	Bidimensional
Two-level serializability	Secuencialidad de dos niveles
Two-phase commit (2PC) protocol	Protocolo de compromiso de dos fases (C2F)
Two-phase locking protocol	Protocolo de bloqueo de dos fases
Two-safe	Dos seguro
Two-very-safe	Dos muy seguro
Typical multiuser system	Sistema multiusuario típico
Typical single-user system	Sistema monousuario típico
Unary	Unaria
Uncommited data	Datos no comprometidos
Uncommited modification	Modificación no comprometida
Undo	Deshacer
Undo operation	Operación undo
Undo phase	Fase deshacer
Undo-list	Lista-deshacer
Unifying model	Modelo de unificación
Union	Unión
Union join	Reunión de unión
Union operation	Operación unión
Union rule	Regla de la unión
Unique	Único
Unique identifier	Identificador único
Unique-role assumption	Suposición de papel único
Unit	Unidad
Universal resource locator (URL)	Localizador universal de recursos
Universel temps coordoné (UTC)	Hora universal coordinada
Unnesting	Desanidamiento
Unsafe	No segura
Unsafe workflow specification	Especificación no segura del flujo de trabajo
Unstructured documents	Documentos no estructurados
Unswizzle	Devolver
Update	De actualización
Update log record	Registro de actualización del registro histórico
Update mode	Modo de actualización
Update transaction	Transacción de actualización
Updates authorization	Autorización de las actualizaciones
URL (Universal resource locator)	Localizador universal de recursos
User	Usuario
User interfaces	Interfaces de usuario
User-coded functions	Funciones codificadas por el usuario

User-defined design constraints	Restricciones de diseño definidas por los usuarios
User-guided data mining	Recopilación de datos dirigida por el usuario
UTC (Universel temps coordoné)	Hora universal coordinada
Utilization	Utilización
Valid time	Tiempo válido
Validation phase	Fase de validación
Validation protocol	Protocolo de validación
Validation scheme	Esquema de validación
Value dependency	Dependencia de valores
Value set	Conjunto de valores
Variable	Variable
Variable-length records	Registros de longitud variable
Vector partitioning	Vector de división
Version vector	Vector de versiones
Vertex	Nodo
Vertical fragmentation	Fragmentación vertical
Video server	Servidor de vídeo
View	Vista
View equivalent	Equivalente en cuanto a vistas
View expansion	Expansión de vistas
View level	Nivel de vistas
View relation	Relación de vistas
View serializability	Secuencialidad en cuanto a vistas
View serializable	Secuenciable en cuanto a vistas
View-maintenance problem	Problema del mantenimiento de las vistas
Virtual memory	Memoria virtual
Virtual record	Registro virtual
Volatile storage	Almacenamiento volátil
Volume	Volumen
WAIS (Wide area information system)	Sistema de información de área amplia
Wait	Esperar
Wait-die scheme	Esquema esperar-morir
Wait-for graph	Grafo de espera
WAL (Write-ahead logging)	Registro histórico de escritura anticipada
WAN (Wide-area network)	Red de área amplia
Weak consistency	Consistencia débil
Weak entity set	Conjunto de entidades débil
Web crawler	Web crawler
Web server	Servidor Web
Where operation	Operación where
Wide area information system (WAIS)	Sistema de información de área amplia
Wide-area network (WAN)	Red de área amplia

Workflow	Flujo de trabajo
Workflow management	Gestión del flujo de trabajo
Workflow Management Coalition	Coalición para la gestión de flujos de trabajo
Workflow specification	Especificación del flujo de trabajo
Workflow-management system	Sistema gestor de flujos de trabajo
World Wide Web (WWW)	World Wide Web
WORM (Write-once, read-many)	Escritura única y lectura múltiple
WORM jukebox	Cambiador de discos WORM
Wound-wait scheme	Esquema herir-esperar
Write phase	Fase de escritura
Write-ahead logging (WAL)	Registro histórico de escritura anticipada
Write-once, read-many (WORM)	Escritura única y lectura múltiple
Wrong	Erróneo
WWW (World Wide Web)	World Wide Web
X/open distributed transaction processing standard	Norma para el procesamiento de transacciones distribuidas X/Open
XML (Extensible Markup Language)	Lenguaje de marcas extensible

Español	Inglés
1FN (Primera forma normal)	First normal form (1NF)
2FN (Segunda forma normal)	Second normal form (2NF)
3FN (Tercera forma normal)	Third normal form (3NF)
4FN (Cuarta forma normal)	Fourth normal form (4NF)
ABD (Administrador de la base de datos)	Database administrator (DBA)
Abortada (transacción)	Aborted
Abstracción	Rollup
Abstracción de datos	Data abstraction
Abstracto	Abstract
Acceso multiclave	Multiple-key access
Acíclico	Acyclic
Administrador de bases de datos (ABD)	Database administrator (DBA)
Advertencia	Alerting
Agente para peticiones de objetos	Object request broquer (ORB)
Agregación	Aggregation
Agrupación	Cluster (*n.*)
Agrupación	Grouping
Agrupamiento	Clustering
Agrupar	Cluster (*v.*)
Aislamiento	Isolation
Ajuste	Tuning
Ajuste del rendimiento	Performance tuning
Álgebra relacional	Relational algebra
Algoritmo de control de concurrencia en árboles B^+	B^+-tree concurrency-control algorithm
Algoritmo de detección de interbloqueos completamente distribuido	Fully distributed deadlock-detection algorithm
Algoritmo de elección	Election algorithm
Algoritmo de mezcla-ordenación externa	External merge-sort algorithm
Algoritmo de recuperación	Recovery algorithm
Algoritmo de reunión paralela	Parallel join algorithm
Algoritmo de reunión por asociación híbrida	Hybrid hash-join algorithm
Algoritmo de reunión por mezcla	Merge-join algorithm
Algoritmo de reunión por mezcla-ordenación	Sort-merge-join algorithm
Algoritmo del ascensor	Elevator algorithm
Algoritmo híbrido de reunión por mezcla	Hybrid merge-join algorithm
Algoritmo luchador	Bully algorithm
Algoritmo para control de la concurrencia para los árboles B^+	Concurrency control B^+-trees algorithm
Algoritmo para el tratamiento de interbloqueos	Deadlock-handling algorithm
Algoritmo para la detección de interbloqueos	Deadlock-detection algorithm
Algoritmo para la evaluación de consultas	Query-evaluation algorithm

Algoritmos de control de concurrencia en árboles B^+	B^+-tree concurrency control algorithms
Algoritmos de evaluación de consultas	Evaluation algorithms queries
Alias	Alias
Almacén de datos	Data warehouse
Almacenado	Stored
Almacenamiento de acceso directo	Direct-access storage
Almacenamiento de acceso secuencial	Sequential-access storage
Almacenamiento de datos	Data storage
Almacenamiento de datos en caché	Data caching
Almacenamiento en cinta	Tape storage
Almacenamiento en conexión	On-line storage
Almacenamiento en disco	Disk storage
Almacenamiento en discos magnéticos	Magnetic-disk storage
Almacenamiento estable	Stable storage
Almacenamiento no volátil	Nonvolatile storage
Almacenamiento óptico	Optical storage
Almacenamiento principal	Primary storage
Almacenamiento secundario	Secondary storage
Almacenamiento sin conexión	Off-line storage
Almacenamiento terciario	Tertiary storage
Almacenamiento volátil	Volatile storage
Ampliabilidad	Scaleup
Ampliabilidad de transacciones	Transaction scaleup
Ampliabilidad lineal	Linear scaleup
Ampliabilidad por lotes	Batch scaleup
Ampliabilidad sublineal	Sublinear scaleup
Análisis estadísticos	Statistical analyses
Anfitrión	Host
Anidamiento	Nesting
Anillo con paso de testigo	Token ring
Any	Any
Aparición del conjunto	Set occurrence
Aplicación informática	Computer application
Aplicaciones de bases de datos	Database applications
Aplicaciones multisistema	Multisystem applications
Apuntar	Point
Árbol B^+	B^+-tree
Árbol con raíz	Rooted tree
Árbol cuadrático	Quadtree
Árbol cuadrático PR	PR quadtree
Árbol de clasificación	Classification tree
Árbol de la base de datos	Database tree

Árbol de operadores	Operator tree
Árbol hijo más a la izquierda	Leftmost-child tree
Árbol k-d	k-d tree
Árbol k-d-B	k-d-B tree
Árbol R	R-tree
Árboles B enlazados	B-link trees
Árboles binarios	Binary trees
Árboles cuadráticos con regiones	Region quadtrees
Arco	Edge
Archivo	File
Archivo de datos	Data archival
Archivo en retícula	Grid file
Archivo secuencial	Sequential file
Archivo secuencial indexado	Index-sequential file
Arpanet	Arpanet
Arquitectura común de agente para peticiones de objetos	Common object request broker architecture (CORBA)
Arquitectura de disco compartido	Shared-disk architecture
Arquitectura de memoria compartida	Shared-memory architecture
Arquitectura jerárquica	Hierarchical architecture
Arquitectura para gestión de objetos	Object management architecture (OMA)
Arquitectura paralela	Parallel architecture
Arquitectura sin compartimiento	Shared-nothing architecture
Arquitecturas distribuidas de memoria virtual	Distributed virtual-memory architecture
Arquitecturas paralelas de bases de datos	Parallel database architectures
Array	Array
Array en retícula	Grid array
Ascendente	Bottom-up
Ascending order	Ascending order
Aserto	Assertion
Asignación	Assignment
Asociación	Hashing
Asociación abierta	Open hashing
Asociación cerrada	Closed hashing
Asociación de funciones	Function hashing
Asociación de índice	Index hashing
Asociación dinámica	Dynamic hashing
Asociación dividida	Partitioned hashing
Asociación estática	Static hashing
Asociación extensible	Extendable hashing
Asociación lineal	Linear hashing
Asociar	Hash
Asumir	Assume

Atasco de cajones	Bucket skew
Atomicidad	Atomicity
Atomicidad ante fallos	Failure-atomicity
Atómico	Atomic
Átomo	Atom
Atributo	Attribute
Atributos compuestos	Composite attributes
Atributos derivados	Derived attributes
Atributos descriptivos	Descriptive attributes
Atributos multivalorados	Multivalued attributes
Atributos nulos	Null attributes
Autonomía local	Local autonomy
Autorización	Authorization
Autorización de índice	Index authorization
Autorización de las actualizaciones	Updates authorization
Autorización de lectura	Read authorization
Autorización de recursos	Resource authorization
Axioma	Axiom
Axiomas de Armstrong	Armstrong's axioms
Bajo coste	Inexpensive
Base	Base
Base de datos (BD)	Database (DB)
Base de datos distribuida homogénea	Homogeneous distributed database
Base de datos en forma normal de dominios y claves (FNDC)	Domain-key normal form (DKNF) database
Base de datos en red	Network database
Base de datos temporal	Temporal database
Bases de datos centralizadas	Centralized databases
Bases de datos CODASYL DBTG 1991	CODASYL DBTG 1991 databases
Bases de datos de hipertexto	Hypertext databases
Bases de datos de Microsoft Access	Microsoft Access databases
Bases de datos distribuidas y heterogéneas	Heterogeneous distributed databases
Bases de datos espaciales	Spatial databases
Bases de datos estadísticas	Statistical databases
Bases de datos geográficas	Geographic databases
Bases de datos jerárquicas	Hierarchical databases
Bases de datos multimedia	Multimedia databases
Bases de datos orientadas a objetos	Object-oriented databases
Bases de datos para diseño	Design databases
Bases de datos para diseño asistido por computadora (CAD)	Computer-aided-design (CAD) databases
Bases de datos portátiles	Mobile databases
Bases de datos relacionales	Relational databases
Bases de datos relacionales orientadas a objetos	Object relational databases

Basura	Garbage
BD (Base de datos)	Database (DB)
Bidimensional	Two-dimensional
Billete	Ticket
Binario	Binary
Bloque	Block
Bloque de desbordamiento	Overflow block
Bloque de memoria intermedia	Buffer block
Bloque físico	Physical block
Bloqueo	Blocking/Lock
Bloqueo compartido	Shared lock
Bloqueo con límite de tiempo	Timeout lock
Bloqueo del índice	Index locking
Bloqueo en modo exclusivo	Exclusive mode lock
Bloqueo exclusivo	Exclusive lock
Bloqueo intencional	Intention lock
Borrado	Deletion
Borrar	Delete
Brazo del disco	Disk arm
Bus	Bus
Búsqueda binaria	Binary search
Búsqueda lineal	Linear search
Búsqueda secuencial	Sequential scan
C2F (Protocolo de compromiso de dos fases)	Two-phase commit (2PC) protocol
C3F (Protocolo de compromiso de tres fases)	Three-phase commit (3PC) protocol
Cabecera	Header
Cabecera de archivo	File header
Cabeza	Head
Cabeza de lectura y escritura	Read-write head
Caché	Cache
CAD (Bases de datos para diseño asistido por computadora)	Computer-aided-design (CAD) databases
CAD de bases de datos	Database CAD
Cadena de desbordamiento	Overflow chaining
Caída	Crash
Caída del sistema	System crash
Caja	Box
Caja de condición	Condition box
Caja límite	Bounding box
Cajón	Bucket
Cajones de desbordamiento	Overflow buckets
Cálculo relacional de dominios	Domain relational calculus
Cálculo relacional de tuplas	Tuple relational calculus

Cambiador automático	Jukebox
Cambiador de cintas	Tape jukebox
Cambiador de discos WORM	WORM jukebox
Cambio de contexto	Context switch
Caminos de acceso	Access paths
Campo puntero	Pointer fields
Campos largos	Long fields
Cancelar	Kill
Capa	Layer
Catálogo del sistema	System catalog
Cauce	Pipeline
Celda	Cell
Centro científico de IBM	IBM Scientific Center
Centro de investigación de Almadén	Almaden Research Center
Centro de investigación T. J. Watson de IBM	IBM T. J. Watson Research Center
Ciclo	Pass
Ciclos falsos	False cycles
Cierre	Closure
Cierre transitivo	Transitive closure
Cifrado	Encryption
Cifrado de clave pública	Public-key encryption
Cifrar	Encrypt
Cintas magnéticas	Magnetic tapes
Clase	Class
Clases de objetos	Object classes
Clasificación	Classification
Clavado	Pinned
Clave	Key
Clave candidata	Candidate key
Clave de búsqueda	Search key
Clave de cifrado	Encryption key
Clave externa	Foreign key
Clave parcial	Partial key
Clave primaria	Primary key
Clave privada	Private key
Clave pública	Public key
Cliente-servidor	Client-server
Coalición para la gestión de flujos de trabajo	Workflow Management Coalition
Código intermedio	Byte code
Códigos Reed-Solomon	Reed-Solomon codes
Coherencia de caché	Cache coherency
Cola	Queue

Colgante	Dangling
Combinar	Combine
Comparación	Comparison
Comparación de conjuntos	Set comparison
Compatible	Compatible
Complejo	Complex
Completo	Complete
Componente (*m.*)	Component
Componente de control de concurrencia	Concurrency-control component
Componente de gestión de transacciones	Transaction-management component
Componente para la gestión de la recuperación	Recovery-management component
Componentes de concurrencia	Concurrency components
Composición de operaciones relacionales	Composition of relational operations
Comprobación de suma	Checksum
Comprometer	Commit
Comprometido	Commited
Compromiso	Commitment
Compuesto	Composite
Computadora anfitriona	Host computer
Computadora portátil	Mobile computer
Comunicar	Call back
Con compromiso de lectura	Read commited
Con conexión	On-line
Conceptual	Conceptual
Concesión	Grant
Concesión de bloqueos	Granting of locks
Concesión de privilegios	Granting of privileges
Concreción	Drill down
Condiciones suficientes	Sufficient conditions
Conectivas lógicas	Logical connectives
Conectividad abierta de bases de datos	Open database connectivity (ODBC)
Conexión continua	Continuous connection
Confianza	Confidence
Configuración de relevo en caliente	Hot-spare configuration
Confluente	Confluent
Conjunto DBTG	DBTG set
Conjunto de entidades	Entity set
Conjunto de entidades de nivel superior	Higher-level entity set
Conjunto de entidades débil	Weak entity set
Conjunto de entidades fuerte	Strong entity set
Conjunto de entrenamiento	Training set
Conjunto de relaciones	Relationship set

Conjunto de valores	Value set
Consejo de rendimiento de procesamiento de transacciones	Transaction processing performance council (TPC)
Conservación de las dependencias	Dependency preservation
Consistencia	Consistency
Consistencia de grado dos	Degree-two consistency
Consistencia de los datos	Data consistency
Consistencia débil	Weak consistency
Constructora	Constructor
Construir	Build
Consulta	Query
Consulta de rangos	Range query
Consulta de vecino más próximo	Nearest-neighbor query
Consulta mediante ejemplos	Query-by-example (QBE)
Consultas con tipos complejos	Complex queries types
Consultas concretas	Point queries
Consultas de ayuda para las decisiones	Decision-support queries
Consultas de proximidad	Nearness queries
Consultas espaciales	Spatial queries
Consultas regionales	Region queries
Contador lógico	Logical counter
Contaminación de los datos	Data pollution
Contiene	Contains
Continente de objetos	Object containment
Contraseña	Password
Control de la concurrencia	Concurrency control
Control de las transacciones	Transaction control
Controlador de disco	Disk controller
Conversiones de bloqueo	Lock conversions
Coordinador copia de seguridad	Backup coordinator
Coordinador de transacciones	Transaction coordinator
Copia en la sombra	Shadow copy
Copia principal	Primary copy
Corrección fuerte	Strong correctness
Correcto	Sound
Correspondencia	Mapping
Correspondencia de cardinalidades	Mapping cardinalities
Correspondencia de objetos a archivos	Objects to files mapping
Correspondencia de uno a uno	One to one mapping
Correspondencia de uno a varios	One to many mapping
Correspondencia de varios a uno	Many to one mapping
Correspondencia de varios a varios	Many to many mapping
Coste de la evaluación paralela de operaciones	Cost of parallel evaluation of operations

Coste de la transmisión de datos	Data transmission cost
Coste mínimo	Minimum cost
Creación de imágenes	Mirroring
Creación de índices de datos espaciales	Indexing spatial data
Creación de índices de documentos	Indexing documents
Creación de sombras	Shadowing
Cuarta forma normal (4FN)	Fourth normal form (4NF)
Cuasisecuencialidad	Quasi-serializability
Cuello de botella	Bottleneck
Cuerpo	Body
Datalog	Datalog
Datos abstractos	Abstract data
Datos de medios continuos	Continuous-media data
Datos de sonido y de vídeo	Audio and video data
Datos de texto	Text data
Datos distribuidos	Distributed data
Datos geográficos	Geographic data
Datos gráficos	Graphical data
Datos isócronos	Isochronous data
Datos multidimensionales	Multidimensional data
Datos no comprometidos	Uncommited data
Datos semiestructurados	Semistructured data
De actualización	Update
De manera contigua	Contiguously
Definición de las estructuras de datos	Data structure definition
Definido por atributo	Attribute-defined
Denominación de datos	Data naming
Dependencia de existencia	Existence dependency
Dependencia de generación de igualdad	Equality-generating dependency
Dependencia de generación de tuplas	Tuple-generating dependency
Dependencia de subconjunto	Subset dependency
Dependencia de valores	Value dependency
Dependencia funcional	Functional dependency
Dependencia multivalorada	Multivalued dependency
Dependencia multivalorada trivial	Trivial multivalued dependency
Dependencias de existencia	Existence dependencies
Dependencias de reunión	Join dependencies
Dependencias funcionales	Functional dependencies
Dependencias transitivas	Transitive dependencies
Depender de	To depend on
Dependiente parcialmente	Partially dependent
Desacoplada	Decoupled

Desanidamiento	Unnesting
Desbordamiento de cajones	Bucket overflow
Desbordamiento de una tabla de asociación	Hash-table overflow
Descendente	Top-down
Descifrar	Decrypt
Descomponer	Decompose
Descomposición	Decomposition
Descomposición con conservación de las dependencias	Dependency-preserving decomposition
Descomposición con pérdida	Lossy decomposition
Descomposición de reunión con pérdida	Lossy-join decomposition
Descomposición de reunión sin pérdida	Lossless-join decomposition
Descomposición que conserva las dependencias	Dependency-preserving decomposition
Descubrimiento de reglas de asociación	Association rules discovery
Deshacer	Undo
Desmarcar	Checkout
Desplazamiento	Offset
Desreferenciación	Dereferencing
Desreferenciar	Dereference
Destructor	Destructor
Detección de fallos	Failure detection
Detección de interbloqueos	Deadlock detection
Devolución	Deswizzling
Devolver	Unswizzle
Diagrama de estructura de datos	Data-structure diagram
Diagrama de la estructura de datos	Data structure diagram
Diagrama entidad-relación	Entity-relationship diagram
Diagrama E-R	E-R diagram
Diagramas de estructura de árbol	Tree-structure diagrams
Diccionario de datos	Data dictionary
Diferencia de conjuntos	Set difference
Diferida	Deferred
Difusión de datos	Data broadcast
Difuso	Fuzzy
Dinámico	Dynamic
Dirección de entrega	Forwarding address
Directorio de datos	Data directory
Disco de cabezas fijas	Fixed-head disk
Disco de registro histórico	Log disk
Disco imagen	Mirror disk
Disco rígido	Hard disk
Discos imagen	Disk mirroring
Discos ópticos	Optical disks

Discriminante	Discriminator
Diseñadores de informes	Report writers
Diseño asistido por computadora	Computed-aided design (CAD)
Diseño conceptual	Conceptual-design
Diseño de bases de datos	Database design
Diseño físico	Physical-design
Diseño lógico	Logical-design
Disparador	Trigger (*n.*)
Disparar	Trigger (*v.*)
Disponibilidad elevada	High availability
Disposición redundante de discos independientes	Redundant array of independent disks (RAID)
Dispositivos cabeza-disco	Head-disk assemblies
Disquetes	Floppy disks
Distribución a nivel de bloque	Block-level striping
Distribución a nivel de bit	Bit-level striping
Distribución de datos	Data striping
Disyunción	Or
Dividir	Partition/ Split
División	Division
División con carga equilibrada	Load-balanced partitioning
División de datos	Data partitioning
División de la red	Network partitioning
División horizontal	Horizontal partitioning
División por asociación	Hash partitioning
División por rangos	Range partitioning
División por turno rotatorio	Round-robin partitioning
División recursiva	Recursive partitioning
Documento	Document
Documentos activos	Active documents
Documentos no estructurados	Unstructured documents
Dominio	Domain
Dominio atómico	Atomic domain
Dos muy seguro	Two-very-safe
Dos seguro	Two-safe
Durabilidad	Durability
Duración	Span
Eficiente	Efficient
Ejecución de los flujos de trabajo	Execution of workflows
Ejecuciones no secuenciables	Nonserializable executions
Ejemplar	Instance
Ejemplar básico	Ground instantation
Ejemplar básico de una regla	Ground instantiation of a rule

Ejemplar de la base de datos	Database instance
Ejemplar de relación	Relation instance
Elección	Election
Elemento ejemplo	Example element
Eliminación de duplicados	Duplicate elimination
Eliminar	Drop
Emplazamiento principal	Primary site
Encaminador	Router
Encaminamiento	Routing
Encaminamiento con varios servidores y un solo servidor	Many-server, single-router routing
Encaminar	Route
Encapsulamiento	Encapsulating
Encapsulamiento de datos	Data encapsulating
Encauzamiento	Pipelining
Encauzamiento bajo demanda	Demand-driven pipelining
Encauzamiento por los productores	Producer-driven pipelining
Enciclopedias interactivas	On-line encyclopedias
Encuestas	Polling
Enfoque centralizado	Centralized approach
Enfoque de varios coordinadores	Multiple-coordinator approach
Enlace	Link
Ensayo lineal	Linear probing
Entidad	Entity
Entidad de procesamiento	Processing entity
Entidad dominante	Dominant entity
Entidad subordinada	Subordinate entity
Entorno distribuido	Distributed environment
Entrada del índice	Index entry
Entrada para construir	Build input
Entrada para probar	Probe input
Equilibrado	Balanced
Equirreunión	Equi-join
Equivalente en cuanto a conflictos	Conflict equivalent
Equivalente en cuanto a vistas	View equivalent
Erróneo	Wrong
Error del sistema	System error
Error lógico	Logical error
Escalas lineales	Linear scales
Escritura única y lectura múltiple	Write-once, read-many (WORM)
Escrituras a ciegas	Blind writes
Escrituras externas observables	Observable external writes
Espacio de intercambio	Swap space

Especialización	Specialization
Especificación	Specification
Especificación de requisitos del usuario	Specification of user requirements
Especificación de requisitos funcionales	Specification of functional requirements
Especificación de seguridad	Security specification
Especificación de una norma para las bases de datos	Database-standard specification
Especificación del flujo de trabajo	Workflow specification
Especificación no segura del flujo de trabajo	Unsafe workflow specification
Esperar	Wait
Esqueletos de tablas	Skeleton tables
Esquema	Schema
Esquema de bases de datos entidad-relación	Entity-relationship database schema
Esquema de control de concurrencia multiversión	Multiversion concurrency-control scheme
Esquema de detección y recuperación de interbloqueos	Deadlock-detention and deadlock-recovery scheme
Esquema de la base de datos	Database schema
Esquema de la empresa	Enterprise schema
Esquema de la relación	Relation schema
Esquema de modificación inmediata	Immediate-modifications scheme
Esquema de ordenación por marcas temporales	Timestamp-ordering scheme
Esquema de ordenación por marcas temporales multiversión	Multiversion timestamp-ordering scheme
Esquema de recuperación	Recovery scheme
Esquema de redundancia P+Q	P+Q redundancy scheme
Esquema de rotación única	Single relation schema
Esquema de sustitución de bloques LRU	LRU block-replacement scheme
Esquema de tiempo límite	Timeout scheme
Esquema de validación	Validation scheme
Esquema esperar-morir	Wait-die scheme
Esquema físico	Physical schema
Esquema herir-esperar	Wound-wait scheme
Esquema lógico	Logical schema
Esquemas de control de concurrencia	Concurrency-control schemes
Esquemas de volcado difuso	Fuzzy dump schemes
Estabilidad del cursor	Cursor stability
Estaciones de apoyo para computadoras portátiles	Mobile support stations
Estado de terminación abortado	Aborted termination state
Estado de terminación aceptable	Acceptable termination state
Estado de terminación aceptable abortado	Aborted acceptable termination state
Estado de terminación aceptable comprometido	Committed acceptable termination state
Estado de terminación comprometido	Commited termination state
Estado de terminación no aceptable	Nonacceptable termination state
Estado de un flujo de trabajo	State of a workflow
Estado inconsistente	Inconsistent state

Estados de ejecución	Execution states
Estados de terminación	Termination states
Estimar	Estimate
Estrategia de extracción inmediata	Toss-immediate strategy
Estrategia de semirreunión	Semijoin strategy
Estrategia para la sustitución de bloques	Block-replacement strategy
Estructura de anillo	Ring structure
Estructura de árbol	Tree structure
Estructura de árbol en memoria	In-memory tree structure
Estructura de archivos	File structure
Estructura de índice de árbol B^+	B^+-tree index structure
Estructura de páginas con ranuras	Slotted-page structure
Ethernet	Ethernet
Evaluación materializada	Materialized evaluation
Evento	Event
Evitación	Avoidance
Expansión de vistas	View expansion
Explícito	Explicit
Exploración	Browsing
Exploración de una relación	Scanning a relation
Exploraciones del índice	Index scans
Explorador de archivo	File scan
Expresión del álgebra relacional	Relational-algebra expression
Expropiar	Preempt
Extensión	Extension
Extensiones de clases	Class extents
Extensiones persistentes	Persistent extensions
Externa (reunión)	Outer
Externo	External
Extracción	Pulling
Factor de escape	Fudge factor
Fallo de disco	Disk failure
Fallo de enlace	Link failure
Fallo de página	Page fault
Fallo del sistema	System failure
Fallo en la transferencia de datos	Data transfer failure
Fallo en transacción	Transaction failure
Fase de crecimiento	Growing phase
Fase de decrecimiento	Shrinking phase
Fase de escritura	Write phase
Fase de lectura	Read phase
Fase de validación	Validation phase

Fase deshacer	Undo phase
Fase rehacer	Redo phase
Fast Path de IMS	IMS Fast Path
Fenómeno fantasma	Phantom phenomenon
Fiabilidad	Reliability
Ficticio	Dummy
Fila	Row
Filas ejemplo	Example rows
Fin-de-registro	End-of-record
Firma digital	Digital signature
Físico	Physical
Flujo de tareas	Task flow
Flujo de trabajo	Workflow
Flujo de trabajo de transacciones	Transaction workflow
FNBC (Forma normal de Boyce-Codd)	Boyce-Codd normal form (BCNF)
FNDC (Forma normal de dominios y claves)	Domain-key normal form (DKNF)
FNRP (Forma normal de reunión por proyección)	Project-join normal form (PJNF)
Forma normal	Normal form
Forma normal de Boyce-Codd (FNBC)	Boyce-Codd normal form (BCNF)
Forma normal de dominios y claves (FNDC)	Domain-key normal form (DKNF)
Forma normal de reunión por proyección (FNRP)	Project-join normal form (PJNF)
Formatos de datos multimedia	Multimedia data formats
Formulario	Form
Fortran	Fortran
Forzar el registro histórico	Log force
Fraccionamiento de costes	Downsizing
Fragmentación	Fragmentation
Fragmentación de datos	Data fragmentation
Fragmentación horizontal	Horizontal fragmentation
Fragmentación mixta	Mixed fragmentation
Fragmentación vertical	Vertical fragmentation
Fragmentar	Fragment
Fragmento	Fragment
Fragmentos y réplicas	Fragment and replicate
Fragmentos y réplicas asimétricos	Asymmetric fragment and replicate
Función de agregación	Aggregation function
Función de agregación count	Count aggregate function
Función de asociación	Hash function
Función de compatibilidad	Compatibility function
Función monótona	Monotonic function
Funciones codificadas por el usuario	User-coded functions
Fundamentos de bases de datos	Database System Concepts

Fusión	Coalescence
Fusionar	Coalesce
GAD (Grafo acíclico dirigido)	Directed acyclic graph (DAG)
Ganancia de velocidad	Speedup
Ganancia de velocidad lineal	Linear speedup
Ganancia de velocidad sublineal	Sublinear speedup
Generalización entidad-relación	Entity-relationship generalization
Gestión de la memoria intermedia	Buffer management
Gestión de transacciones	Transaction management
Gestión del flujo de trabajo	Workflow management
Gestor de almacenamiento	Storage manager
Gestor de colas	Queue manager
Gestor de control de concurrencia	Concurrency-control manager
Gestor de la memoria intermedia	Buffer manager
Gestor de recursos	Resource manager
Gestor de transacciones locales	Local transaction manager
Gopher	Gopher
Grado	Degree
Grado de relevancia	Degree of relevance
Grado de salida	Fan-out
Grafo	Graph
Grafo acíclico dirigido (GAD)	Directed acyclic graph (DAG)
Grafo construido	Constructed graph
Grafo de autorización	Authorization graph
Grafo de dependencia	Dependency graph
Grafo de espera	Wait-for graph
Grafo de espera local	Local wait-for graph
Grafo de histograma	Histogram graph
Grafo de la base de datos	Database graph
Grafo de precedencia	Precedence graph
Grafo de precedencia etiquetado	Labeled precedence graph
Grafo real	Real graph
Granularidad	Granularity
Granularidad de los bloqueos	Granularity locks
Granularidad fina	Fine granularity
Granularidad gruesa	Coarse granularity
Granularidad múltiple	Multiple granularity
Grupo	Group
Grupo conjunto de expertos en imágenes	Joint picture experts group (JPEG)
Grupo de acceso SQL	SQL Access Group
Grupo de expertos en películas	Motion picture experts group (MPEG)
Grupo de gestión de bases de datos de objetos	Object database management group (ODMG)

Grupo de gestión de objetos	Object management group (OMG)
Grupo de trabajo sobre bases de datos	Database Task Group
Hecho	Fact
Herencia	Inheritance
Herencia de atributos	Attribute inheritance
Herencia de atributos entidad-relación	Entity-relationship attribute inheritance
Herencia múltiple	Multiple inheritance
Heurística	Heuristic
Hijo (o miembro)	Child (or member)
Hipertexto	Hypertext
Histograma	Histogram
Hora universal coordinada	Universel temps coordoné (UTC)
HyperCard	HyperCard
Idempotente	Idempotent
Identificador de objeto	Object identifier (OID)
Identificador de objeto	OID (Objects identifier)
Identificador físico de objeto	Physical object identifier
Identificador lógico de objeto	Logical object identifier
Identificador único	Unique identifier
Identificar	Identify
Id-tupla	Tuple-id
Igualdad	Equality
Igualdad basada en la clave	Equality on key
Igualdad basada en un atributo no clave	Equality on nonkey
Impaciente	Eager
Implementación	Implementation
Implementar	Implement
Implicadas lógicamente	Logically implied
Implícito	Implicit
Inanición	Starvation
Inclusión innecesaria	False positive
Inconsistencia de datos	Data inconsistency
Incorporado	Embedded
Independencia de datos	Data independence
Independiente	Independent
Índice	Index
Índice asociativo	Hash index
Índice compuesto	Composite index
Índice con agrupación	Clustering index
Índice de árbol B	B-tree index
Índice de árbol B^+	B^+-tree index
Índice de informes técnicos de informática	Computer Science Technical Report Index

Índice de organización de archivos secuenciales	Sequential file organization index
Índice del árbol B^+	Index B^+-tree
Índice del texto completo	Full-text index
Índice denso	Dense index
Índice disperso	Sparse index
Índice invertido	Inverted index
Índice multinivel	Multilevel index
Índice ordenado	Ordered index
Índice primario	Primary index
Índice secundario	Secondary index
Índice sin agrupación	Nonclustering index
Inferir	Infer
Información geométrica	Geometric information
Informática portátil	Mobile-computing
Informe de invalidación	Invalidation report
Ingeniería del software asistida por computadora	CASE (Computer-aided software engineering)
Ingeniería del software asistida por computadora	Computer-aided software engineering (CASE)
Iniciar	Start
Inicio	Startup
Inmediata	Immediate
Inserción	Insertion/ Pushing
Instituto de ingenieros eléctricos y electrónicos	Institute for electrical and electronical engineers (IEEE)
Instituto de normalización nacional americano	American national standards institute (ANSI)
Instrucción	Sentence
Instrucción revoke	Revoke statement
Integridad	Integrity
Integridad de las bases de datos	Database integrity
Integridad de los datos	Data integrity
Integridad referencial	Referential integrity
Integridad referencial en el lenguaje SQL	SQL language referencial integrity
Interactivo	On-line
Interbloqueo	Deadlock
Interconexión de pequeños sistemas informáticos	Small computer-system interconnect (SCSI)
Interconexión hipercubo	Hypercube interconnection
Interfaces de usuario	User interfaces
Interfaz (*f.*)	Interface
Interfaz de bases de datos para arquitecturas de aplicación a sistemas	System application architecture database interface (SAA-SQL)
Interfaz de programación de aplicaciones	Application programming interface (API)
Interfaz del nivel de llamadas	Call-level interface (CLI)
Interfaz gráfica de usuario	Graphical user interface
Interfaz para formularios	Form interface

Interfaz RPC para transacciones	Transactional RPC interface
Interferencia	Interference
Interna (reunión)	Inner
Internet	Internet
Interno	Internal
Intersección	Intersection
Intersección de conjuntos	Set intersection
Intervalo	Interval
Invocar un método	Invoke a method
Iterador	Iterator
Java	Java
Jerarquía	Hierarchy
Jerarquía de clasificación	Classification hierarchy
Jerarquía de continentes	Containment hierarchy
L4G (Lenguaje de cuarta generación)	Fourth-generation language (4GL)
Laboratorio de Investigación de San José	San José Research Center
LDD (Lenguaje de definición de datos)	Data-definition language (DDL)
Lectura repetible	Repeatable read
Legal	Legal
Lenguaje anfitrión	Host language
Lenguaje de consulta incorporado	Embedded query language
Lenguaje de consultas	Query language
Lenguaje de consultas temporales	Temporal query language
Lenguaje de control de trabajos	Job control language (JCL)
Lenguaje de cuarta generación (L4G)	Fourth-generation language (4GL)
Lenguaje de definición de datos (LDD)	Data-definition language (DDL)
Lenguaje de definición de objetos	Object definition language (ODL)
Lenguaje de descripción de interfaces	Interface description language (IDL)
Lenguaje de manipulación de datos (LMD)	Data-manipulation language (DML)
Lenguaje de marcas de hipertexto	Hypertext markup language (HTML)
Lenguaje de marcas de texto	Text-markup language
Lenguaje de marcas extensible	XML (Extensible Markup Language)
Lenguaje de programación persistente	Persistent programming language
Lenguaje estructurado de consultas	Structured query language (SQL)
Lenguaje incorporado	Embedded language
Lenguaje no procedimental	Nonprocedural language
Lenguaje normalizado de marcas generalizado	Standard generalized markup language (SGML)
Lenguaje procedimental	Procedural language
Lenguaje SQL Gupta	Gupta SQL languages
Lenguaje SQL incorporado	Embedded SQL language
Liberación de bloqueos	Lock deescalation
Línea	Line

Lista de atributos	Attribute list
Lista de privilegios	Privilege list
Lista enlazada	Linked list
Lista libre	Free list
Lista-deshacer	Undo-list
Lista-rehacer	Redo-list
Literal negativo	Negative literal
Literal positivo	Positive literal
LMD (Lenguaje de manipulación de datos)	Data-manipulation language (DML)
Localizador universal de recursos	Universal resource locator (URL)
Lotus Notes	Lotus Notes
Llamada a procedimientos remotos	Remote-procedure-call (RPC)
Llamada a procedimientos remotos de transacciones	Transactional remote procedure call
Manipulación de datos	Data manipulation
Máquina de bases de datos de Teradata	Teradata database machine
Máquina de grano fino	Fine-grain machine
Máquina masivamente paralela	Massively parallel machine
Máquinas paralelas de grano fino	Fine-granularity parallel machines
Marca temporal	Time stamp
Marcar	Checkin
Materialización	Materialization
Materializar	Materialize
Mecanismo de llamadas a procedimientos remotos	Remote-procedure-call mechanism
Media armónica	Harmonic mean
Memoria compartida	Shared memory
Memoria *flash*	Flash memory
Memoria intermedia	Buffer
Memoria intermedia con traducción anticipada	Translation lookaside buffer (TLB)
Memoria intermedia de disco	Disk buffer
Memoria intermedia de la base de datos	Database buffer
Memoria no volátil de acceso aleatorio (RAM no volátil)	Nonvolatile random-access memory (nonvolatile RAM)
Memoria principal	Main memory
Memoria sólo de lectura en disco compacto	Compact-disk read-only memory (CD-ROM)
Memoria sólo de lectura programable y borrable eléctricamente	Electrically erasable programmable read-only memory (EEPROM)
Memoria virtual	Virtual memory
Memoria virtual distribuida	Distributed virtual-memory
Mensaje	Message
Metadatos	Metadata
Método	Method
Mezcla de *n* vías	*N*-way merge
Mezcla-ordenación externa	External sort-merge

Mezcla-ordenación paralela externa	Parallel external sort-merge
Mezclar	Merge
Microcomputadora	Set-top box computer
Miembro (o hijo)	Member (or child)
Modelo de datos	Data model
Modelo de datos entidad-relación	Entity-relationship data model
Modelo de datos físicos	Physical data model
Modelo de datos orientado a objetos	Object-oriented data model
Modelo de disco compartido	Shared-disk model
Modelo de memoria por marcos	Frame-memory model
Modelo de navegación	Navigational model
Modelo de operador de intercambio	Exchange-operator model
Modelo de proceso por cliente	Process-per-client model
Modelo de red	Network model
Modelo de servidor único	Single-server model
Modelo de simulación de rendimiento	Performance-simulation model
Modelo de transacciones distribuidas	Distributed transaction model
Modelo de unificación	Unifying model
Modelo de varios servidores y un solo encaminador	Many-server, single-router model
Modelo de varios servidores y varios encaminadores	Many-server, many-router model
Modelo entidad-relación	Entity-relationship model
Modelo E-R (entidad-relación)	E-R model (Entity-relationship model)
Modelo jerárquico	Hierarchical model
Modelo orientado a objetos	Object-oriented model
Modelo relacional	Relational model
Modelo relacional anidado	Nested relational model
Modelo suceso-condición-acción	Event-condition-action model
Modelos de datos relacionales	Relational data models
Modelos de datos relacionales orientados a objetos	Object-relational data models
Modelos lógicos basados en objetos	Object-based data models
Modelos lógicos basados en registros	Record-based data models
Modelos relacionales orientados a objetos	Object relational models
Módem	Modem
Modificación de bases de datos	Modification of databases
Modificación diferida	Deferred modification
Modificación no comprometida	Uncommited modification
Modo de actualización	Update mode
Modo de bloqueo compartido	Shared mode locks
Modo de bloqueo intencional	Intention lock mode
Modo de transferencia asíncrono (MTA)	Asynchronous transfer mode (ATM)
Modo intencional-compartido (IC)	Intention-shared (IS) mode
Modo intencional-exclusivo (IX)	Intention-exclusive (IX) mode

Modo intencional-exclusivo y compartido (IXC)	Shared and intention-exclusive (SIX) mode
Módulo	Module
Monitor CICS	CICS monitor
Monitor de teleprocesamiento	Teleprocessing monitor
Monitor Encina	Encina monitor
Monitor IMS	IMS monitor
Monitores RTR	RTR monitors
Monótona	Monotonic
Motor de ejecución de consultas	Query-execution engine
MTA (Modo de transferencia asíncrono)	Asynchronous transfer mode (ATM)
Multiconjunto	Multiset
Multienhebramiento	Multithreading
Multinivel	Multilevel
Multiprogramación	Multiprogramming
Multitarea	Multitasking
Multivalorado	Multivalued
Multiversión	Multiversion
Nivel conceptual	Conceptual level
Nivel de vistas	View level
Nivel físico	Physical level
Nivel lógico	Logical level
Nivel más alto	Higher-level
Nivel más bajo	Lower-level
Niveles de RAID	RAID levels
No conocido	Not known
No segura	Unsafe
Nodo	Vertex (graph)/Node (network)
Nodo remoto copia de seguridad	Remote backup site
Nodo secundario	Secondary site
Norma	Standard
Norma anticipativa	Anticipatory standard
Norma de cifrado de datos	Data encryption standard (DES)
Norma de facto	De facto standard
Norma del lenguaje SQL	SQL language standard
Norma formal	Formal standard
Norma MPEG-1	MPEG-1 standard
Norma MPEG-2	MPEG-2 standard
Norma para el procesamiento de transacciones distribuidas X/Open	X/open distributed transaction processing standard
Norma reaccionaria	Reactionary standard
Normalización	Normalization
Nulo	Null

Número de bloque	Block number
Número de transferencias de bloques de disco	Number of block transfers from disk
Object SQL	Object SQL (OSQL)
Objeto	Object
Objeto complejo	Complex object
Objeto compuesto	Composite object
Objeto en binario de gran tamaño	Binary large object (blob)
Observar	Monitor
Omisión incorrecta	False drop
Operación append	Append operation
Operación asignación	Assignment operation
Operación by	By operation
Operación de agregación	Aggregation operation
Operación delete	Delete operation
Operación diferencia de conjuntos	Set-difference operation
Operación división	Division operation
Operación except	Except operation
Operación from	From operation
Operación grant	Grant operation
Operación group by	Group by operation
Operación intersección	Intersection operation
Operación intersect	Intersect operation
Operación join	Join operation
Operación lógica	Logical operation
Operación lookup	Lookup operation
Operación producto cartesiano	Cartesian-product operation
Operación proyección	Project operation
Operación proyección generalizada	Generalized-projection operation
Operación range of	Range of operation
Operación redo	Redo operation
Operación relacional	Relational operation
Operación rename	Rename operation
Operación replace	Replace operation
Operación select	Select operation
Operación undo	Undo operation
Operación unión	Union operation
Operación where	Where operation
Operaciones de cadena	String operations
Operaciones de conjuntos	Set operations
Operador data-cube	Data-cube operator
Operador de agregación	Aggregation operator
Optimización basada en el coste	Cost-based optimization

Optimización con tableaux	Tableau optimization
Optimización de consultas	Query optimization
Oracle 7	Oracle 7
Orden	Command
Orden de secuencialidad	Serializability order
Orden descendente	Descending order
Orden lexicográfico	Lexicographic ordering
Orden más significativo	Big-endian form
Orden menos significativo	Little-endian form
Ordenación externa	External sorting
Ordenación paralela	Parallel sort
Ordenación rápida	Quicksort
Ordenación topológica	Topological sorting
Órdenes de reunión en profundidad por la izquierda	Left-deep join orders
Organización asociativa de archivos	Hashing file organization
Organización de archivos	File organization
Organización de archivos con árboles B^+	B^+-tree file organization
Organización de archivos en agrupaciones	Clustering file organization
Organización de archivos en montículo	Heap file organization
Organización de códigos de corrección de errores tipo memoria	Memory-style error-correcting-code organization (ECC)
Organización de paridad con bits entrelazados	Bit-interleaved parity organization
Organización de paridad con bloques entrelazados	Block-interleaved parity organization
Organización internacional de normalización	International standards organization (ISO)
Organizar lógicamente los documentos	Organize documents logically
Padre (o propietario)	Parent (or owner)
Página	Page
Página inicial	Home page
Paginación en la sombra	Shadow paging
Palabra clave	Keyword
Palabras de parada	Stop words
Papel	Role
Paralelismo de datos	Data parallelism
Paralelismo de E/S	I/O parallelism
Paralelismo de encauzamiento	Pipelined parallelism
Paralelismo de grano grueso	Coarse-granularity parallelism
Paralelismo de particiones	Partitioned parallelism
Paralelismo en consultas	Intraquery parallelism
Paralelismo en operaciones	Intraoperation parallelism
Paralelismo entre consultas	Interquery parallelism
Paralelismo entre operaciones	Interoperation parallelism
Paralelismo independiente	Independent parallelism
Parámetro	Parameter

Parámetros ajustables	Tunable parameters
Parcial	Partial
Parte visible al usuario	Front-end
Partición	Partition (*n.*)
Participación	Participation
Pascal	Pascal
Paso	Step
Paso de mensaje	Sending a message
Pathway	Pathway
Perdido	Missing
Persistencia de los objetos	Persistence of objects
Pertenencia al conjunto	Set membership
Pestillo	Latch
Pista	Track
Plan de ejecución de la consulta	Query-execution plan
Plan de evaluación de la consulta	Query-evaluation plan
Planificación	Scheduling
Planificación del brazo del disco	Disk-arm-scheduling
Planificación no recuperable	Nonrecoverable schedule
Planificaciones de conflictos	Conflict schedules
Planificaciones recuperables	Recoverable schedules
Planificaciones sin cascada	Cascadeless schedules
Planificar	Schedule
Plantilla	Template
Plato	Platter
Población	Population
Polígonos cerrados	Closed polygons
Por líneas	Raster
Posibilidad	Possibility
Posibilidad de sustitución	Substitutability
Postgres	Postgres
PostScript	PostScript
PowerBuilder	PowerBuilder
Práctico	Convenient
Precedencia de grafos	Graph precedence
Precedencia de grafos etiquetados	Labeled graph precedence
Preceder	Precede
Precio por TPS	Price per TPS
Precisión	Precision
Precompilador del LMD	DML precompiler
Preextracción	Prefetching
Prefijo común de la función de asociación	Common hash prefix

Preservar la equivalencia	Preserve equivalence
Prevención de interbloqueos	Deadlock prevention
Primera forma normal (1FN)	First normal form (1NF)
Primitivas de evaluación	Evaluation primitives
Primo	Prime
Privilegio	Privilege
Privilegio de referencias	Reference privilege
Privilegios de referencia del lenguaje SQL	SQL language reference privilege
Probar	Probe
Problema de bloqueo	Blocking problem
Problema de coherencia caché	Cache-coherency problem
Problema del mantenimiento de las vistas	View-maintenance problem
Problemas NP-completos	NP-complete problems
Procesador de consultas	Query processor
Procesamiento	Processing
Procesamiento de consultas	Query processing
Procesamiento de transacciones	Transaction processing (TP)
Procesamiento distribuido de consultas	Distributed query processing
Procesamiento en conexión analítico	On-line analytical processing (OLAP)
Procesamiento en conexión de transacciones	On-line transaction processing (OLTP)
Proceso	Process
Procesos del servidor para varias aplicaciones	Multiple application-server processes
Productividad	Throughput
Producto cartesiano	Cartesian product
Programa	Program
Programación orientada a objetos	Object oriented programming (OOP)
Programas de aplicación	Application programs
Prolog	Prolog
Propiedades ACID	ACID properties
Propietario	Own
Propietario (o padre)	Owner (or parent)
Protocolo basado en el bloqueo	Lock protocol
Protocolo basado en grafos	Graph-based protocol
Protocolo basado en la marca temporal	Timestamp-based protocol
Protocolo de árbol	Tree protocol
Protocolo de bloqueo	Locking protocol
Protocolo de bloqueo de dos fases	Two-phase locking protocol
Protocolo de bloqueo de dos fases multiversión	Multiversion two-phase locking protocol
Protocolo de bloqueo de granularidad múltiple	Multiple-granularity locking protocol
Protocolo de bloqueo del árbol	Tree-locking protocol
Protocolo de bloqueo estricto de dos fases	Strict two-phase locking protocol
Protocolo de bloqueo riguroso de dos fases	Rigorous two-phase locking protocol

Protocolo de bosque	Forest protocol
Protocolo de compromiso	Commit protocol
Protocolo de compromiso de dos fases (C2F)	Two-phase commit (2PC) protocol
Protocolo de compromiso de tres fases (C3F)	Three-phase commit (3PC) protocol
Protocolo de control de la concurrencia	Concurrency-control protocol
Protocolo de fallo del coordinador	Coordinator-failure protocol
Protocolo de interfaz cliente-servidor	Client-server interface protocol
Protocolo de lectura global	Global-read protocol
Protocolo de lectura local	Local-read protocol
Protocolo de lectura y escritura global y lectura local	Global-read-write/local-read protocol
Protocolo de ordenación por marcas temporales	Timestamp-ordering protocol
Protocolo de prevención de interbloqueos	Deadlock-prevention protocol
Protocolo de transferencia de archivos	File transfer protocol (FTP)
Protocolo de validación	Validation protocol
Protocolo global de lectura y escritura	Global-read-write protocol
Protocolo para transferencia de hipertexto	HyperText transfer protocol (HTTP)
Protocolo sesgado	Biased protocol
Proyección	Projection
Proyección generalizada	Generalized-projection
Proyección temporal	Temporal projection
Proyectar	Project
Prueba de rendimiento	Performance benchmark
Prueba de secuencialidad	Testing for serializability
Puesto	Mart
Puntero	Pointer
Puntero colgante	Dangling pointer
Puntero oculto	Hidden pointers
Puntero persistente	Persistent pointer
Punteros internos de memoria	In-memory pointers
Punto de bloqueo	Lock point
Punto de revisión	Checkpoint
Punto de revisión difuso	Fuzzy checkpoint
Punto fijo	Fixpoint
Punto fijo del lenguaje Datalog	Datalog language fixpoint
Quel	Quel
Rango de las consultas	Query range
Raro	Extraneous
RDSI (Red digital de servicios integrados)	Integrated services digital network (ISDN)
Reasignación de los sectores dañados	Remapping of bad sectors
Recogida de basura	Garbage collection
Recopilación de datos	Data mining
Recopilación de datos dirigida por el usuario	User-guided data mining

Recubrimiento canónico	Canonical cover
Recuperabilidad	Recoverability
Recuperabilidad de elementos de datos de gran tamaño	Large data items recoverability
Recuperabilidad de planificaciones	Recoverability schedule
Recuperación	Recall/Recovery
Recuperación al reiniciar	Restart recovery
Recuperación basada en la semejanza	Similarity-based retrieval
Recuperación de fallos	Failure recovery
Recuperación de fallos basada en el registro histórico	Log-based failure recovery
Recuperación de información basada en palabras clave	Keyword-based information retrieval
Recuperación de interbloqueos	Deadlock recovery
Recuperación de la información	Information retrieval
Recuperar	Recover
Recursividad	Recursion
Recursividad del lenguaje Datalog	Datalog language recursion
Recursivo	Recursive
Recurso de gestión de consultas	Query management facility (QMF)
Red conectada parcialmente	Partially connected network
Red de área amplia	Wide-area network (WAN)
Red de área local	Local-area network (LAN)
Red de computadoras	Computer network
Red de interconexión	Interconnection network
Red digital de servicios integrados (RDSI)	Integrated services digital network (ISDN)
Redes informáticas de larga distancia	Long-distance computer networks
Redistribuir	Redistribute
Redundancia	Redundancy
Redundancia de datos	Data redundancy
Referencia	Reference
Registro	Record
Registro de actualización del registro histórico	Update log record
Registro de compromiso del registro histórico	Commit log record
Registro del índice	Index record
Registro físico del registro histórico	Physical log record
Registro histórico	Log
Registro histórico con memoria intermedia	Log-record buffering
Registro histórico de escritura anticipada	Write-ahead logging (WAL)
Registro histórico físico	Physical logging
Registro histórico lógico	Logical logging
Registro rehacer del registro histórico	Redo log record
Registro virtual	Virtual record
Registros de longitud fija	Fixed-length records
Registros de longitud variable	Variable-length records

Regla	Rule
Regla de escritura de Thomas	Thomas' write rule
Regla de la aumentatividad	Augmentation rule
Regla de la aumentatividad multivalorada	Multivalued augmentation rule
Regla de la complementariedad	Complementation rule
Regla de la descomposición	Decomposition rule
Regla de la diferencia	Difference rule
Regla de la fusión	Coalescence rule
Regla de la intersección	Intersection rule
Regla de la pseudotransitividad	Pseudotransitivity rule
Regla de la reflexividad	Reflexivity rule
Regla de la transitividad	Transitivity rule
Regla de la transitividad multivalorada	Multivalued transitivity rule
Regla de la unión	Union rule
Regla de la unión multivalorada	Multivalued union rule
Regla de réplicas	Replication rule
Reglas activas	Active rules
Reglas completas	Complete rules
Reglas correctas	Sound rules
Reglas de asociación	Association rules
Reglas de Datalog	Datalog rules
Reglas de equivalencia	Equivalence rules
Reglas del lenguaje Datalog	Datalog language rules
Rehacer	Redo
Relación	Relation/Relationship
Relación bitemporal	Bitemporal relation
Relación de identificación	Identifying relationship
Relación de vistas	View relation
Relación instantánea	Snapshot relation
Relación temporal	Temporal relation
Relacional anidado	Nested relational
Relaciones de consultas	Queries relations
Relaciones derivadas	Derived relations
Relevo	Handoff
Relevo del control	Handoff of control
Reloj del sistema	System clock
Reloj lógico	Logical clock
Remitir	Submit
Rendimiento	Performance
Rendimiento de reconstrucción	Rebuild performance
Renombramiento	Renaming
Renombrar	Rename

Reorganizar	Reorganize
Repetición de información	Repetition of information
Repetición de la historia	Repeating history
Réplica completa	Full replication
Réplica de datos	Data replication
Representación en cadenas de bytes	Byte-string representation
Representación en longitud fija	Fixed-length representation
Representación tabular	Tabular representation
Requisito	Requirement
Rescate	Swizzling
Rescate de punteros	Pointer swizzling
Rescate hardware	Hardware swizzling
Rescate software	Software swizzling
Resolución	Resolution
Respuesta	Response
Restricción	Restriction
Restricción de completitud	Completeness constraint
Restricción de integridad referencial	Referential integrity constraint
Restricción general	General constraint
Restricción sobre el carácter disjunto	Disjointness constraint
Restricciones con condiciones definidas	Conditions-defined constraints
Restricciones de autorización	Authorization restrictions
Restricciones de completitud	Constraints completeness
Restricciones de consistencia	Consistency constraints
Restricciones de diseño	Design constraints
Restricciones de diseño definidas por las condiciones	Condition-defined design constraints
Restricciones de diseño definidas por los atributos	Attribute-defined design constraints
Restricciones de diseño definidas por los usuarios	User-defined design constraints
Restricciones de dominio	Domain constraints
Restricciones de integridad	Integrity constraints
Restricciones de integridad referencial	Referencial-integrity constraints
Resumen	Summary
Retículo	Lattice
Retirada en cadena	Cascading of the revoke
Retroceder	Rollback (*v.*)
Retroceso	Rollback (*n.*)
Retroceso de la transacción	Transaction rollback
Retroceso en cascada	Cascading rollback
Reunión	Join
Reunión con fragmentos y réplicas	Fragment-and-replicate join
Reunión con pérdida	Lossy join
Reunión cruzada	Cross join

Reunión de banda	Band join
Reunión de unión	Union join
Reunión en bucle anidado	Nested-loop join
Reunión en bucle anidado indexada	Indexed nested-loop join
Reunión encauzada	Pipelined join
Reunión espacial	Spatial join
Reunión externa	Outer-join
Reunión externa completa	Full outer join
Reunión externa por la derecha	Right outer join
Reunión externa por la izquierda	Left outer join
Reunión fraccionada	Join partitioning
Reunión natural	Natural join
Reunión paralela	Parallel join
Reunión por mezcla-ordenación	Sort-merge join
Reunión por proyección	Project join
Reunión sin pérdida	Lossless join
Reunión temporal	Temporal join
Reuniones dependientes	Dependent joins
Robo de información	Theft of information
Robustez	Robustness
Saga	Saga
Salida forzada	Forced output
Satisfacer	Satisfy
Se cumple en *R*	Holds on *R*
Sector	Sector
Secuencia	Run
Secuenciable en cuanto a conflictos	Conflict serializable
Secuenciable en cuanto a vistas	View serializable
Secuencial	Serial
Secuencialidad	Serializability
Secuencialidad de dos niveles	Two-level serializability
Secuencialidad en cuanto a conflictos	Conflict serializability
Secuencialidad en cuanto a vistas	View serializability
Segunda forma normal (2FN)	Second normal form (2NF)
Seguridad	Safety
Seguridad de la base de datos	Database security
Seguridad en el nivel de la red	Network-level security
Selección	Selection
Selección conjuntiva	Conjunctive selection
Selección conjuntiva mediante la intersección de identificadores	Conjunctive selection by intersection of identifiers
Selección conjuntiva utilizando un índice	Conjunctive selection using one index
Selección conjuntiva utilizando un índice compuesto	Conjunctive selection using composite index

Selección del plan de acceso	Access-plan-selection
Selección disyuntiva	Disjunctive selection
Selección disyuntiva mediante la unión de identificadores	Disjunctive selection by union of identifiers
Selección temporal	Temporal selection
Seleccionar	Select
Selectividad	Selectivity
Semántica de estratificación modular	Modular-stratification semantics
Semántica de un programa	Semantics of a program
Semántica de una regla	Semantics of a rule
Semejante	Similar
Semirreunión	Semi join
Servidor de nombres	Name server
Servidor de vídeo	Video server
Servidor Web	Web server
Sesgar	Skew
Sesgo	Skew
Sesgo de ejecución	Execution skew
Sesgo de la división	Partition skew
Sesgo de la división	Skew partitioning
Sesgo de los valores	Attribute-value skew
Sesgo de los valores de los atributos	Attribute value skew
SGBD (Sistema gestor de bases de datos)	Database management system (DBMS)
Simula	Simula
Sin compromiso de lectura	Read uncommited
Sin conexión	Off-line
Sinónimo	Synonym
Sintetizar	Synthesize
Sistema ampliable	Escalable system
Sistema con múltiples bases de datos	Multidatabase system
Sistema de archivos	File system
Sistema de archivos basado en el registro histórico	Log-based file system
Sistema de bases de datos activas	Database active system
Sistema de bases de datos en memoria principal	Main-memory database system
Sistema de bases de datos Illustra	Illustra database system
Sistema de bases de datos Ingres	Ingres database system
Sistema de colas	Queue system
Sistema de disco compartido	Shared-disk system
Sistema de información de área amplia	Wide area information system (WAIS)
Sistema de localización global	Global positioning system (GPS)
Sistema de procesamiento de archivos	File-processing system
Sistema de tiempo real	Real-time system
Sistema de usuario único	Single user system

Sistema Gamma	Gamma system
Sistema gestor de bases de datos (SGBD)	Database management system (DBMS)
Sistema gestor de flujos de trabajo	Workflow-management system
Sistema gestor de información de IBM	Information management system (IMS)
Sistema Grace	Grace system
Sistema informático	Computer system
Sistema ininterrumpido	Non-stop system
Sistema masivamente paralelo	Massively parallel system
Sistema monousuario típico	Typical single-user system
Sistema multiprocesador de memoria compartida	Shared-memory multiprocessor system
Sistema multiusuario	Multiuser systems
Sistema multiusuario típico	Typical multiuser system
Sistema paralelo de bases de datos	Parallel database system
Sistema paralelos	Parallel system
Sistema relacional	Relational system
Sistema relacional orientados a objetos	Object-relational system
Sistema servidor	Server system
Sistema servidor de consultas	Query-server system
Sistema servidor de transacciones	Transaction server system
Sistema subyacente	Back-end
Sistemas clientes	Client systems
Sistemas de ayuda para las decisiones	Decision-support systems
Sistemas de bases de datos cliente-servidor	Client-server database systems
Sistemas de bases de datos multiversión	Multiversion database systems
Sistemas de colas	Queueing systems
Sistemas de descubrimiento del conocimiento	Knowledge-discovery systems
Sistemas de información geográfica	Geographic information systems
Sistemas de información para oficinas	Office information systems (OIS)
Sistemas de transacciones de alto rendimiento	High-performance transaction systems
Sistemas de visualización de datos	Data-visualization systems
Sistemas distribuidos de bases de datos	Distributed database systems
Sistemas distribuidos de hipertexto	Distributed hypertext systems
Sistemas distribuidos de información	Distributed information systems
Sistemas distribuidos de información en Internet	Distributed Internet information systems
Sistemas distribuidos de información Gopher	Distributed Gopher information systems
Sistemas distribuidos de información World Wide Web	Distributed World Wide Web information systems
Sistemas distribuidos y heterogéneos de bases de datos	Heterogeneous distributed database systems
Sistemas heredados	Legacy systems
Sistemas hipermedia	Hypermedia systems
Sistemas monótonos	Monotonic systems
Sistemas RAID	Redundant array of inexpensive disks systems
Sistemas relacionales orientados a objetos	Object relational systems

Sistemas remotos de copia de seguridad	Remote backup systems
Sistemas servidores de consultas	Queries server systems
Sistemas servidores de datos	Data-server systems
Sistemas servidores de transacciones	Transaction-server systems
Smalltalk	Smalltalk
Sobrecarga	Overloading
Solapa	Overlaps
Solicitud	Request
Sólo de lectura	Read-only
Soporte	Support
Subárboles disjuntos	Disjoint subtrees
Subclase	Subclass
Subconsulta	Subquery
Subesquema	Subschema
Subexpresiones comunes	Common subexpressions
Subsistema de recuperación de caídas	Crash-recovery subsystem
Suceso	Event
Suceso disparador	Triggering event
Superclase	Superclass
Superclave	Superkey
Superusuario	Superuser
Suponer aborto	Presume abort
Suponer compromiso	Presume commit
Suposición de papel único	Unique-role assumption
Supresión de índices	Drop indices
Supuesto de fallo-parada	Fail-stop assumption
System R	System R
Tabla	Table
Tabla cruzada	Cross-tab
Tabla de entradas cruzadas	Cross-tabulation
Tabla de páginas	Page table
Tabla de páginas actual	Current page table
Tabla de páginas sombra	Shadow page table
Tablas combinadas	Combined tables
Tableau	Tableau
Tándem	Tandem
Tarea	Task
Técnica de actualización inmediata	Immediate-update technique
Técnica de bloqueo de índice	Index-locking technique
Técnica de compromiso en grupo	Group-commit technique
Técnicas de asociación dinámica	Dynamic hashing techniques
Técnicas de organización de archivos	File organization techniques

Teoría de colas	Queue theory
Tercera forma normal (3FN)	Third normal form (3NF)
Terminación con éxito	Successful completion
Terminado	Terminated
Terminal	Terminal
Teselación por triangulación	Triangulation
Tiempo de acceso	Access time
Tiempo de búsqueda	Seek time
Tiempo de inicio	Startup time
Tiempo de latencia	Latency time
Tiempo de latencia rotacional	Rotational latency time
Tiempo de servicio	Service time
Tiempo de transacción	Transaction time
Tiempo límite	Deadline
Tiempo medio de búsqueda	Average seek time
Tiempo medio de latencia	Average latency time
Tiempo medio de reparación	Mean time to repair
Tiempo medio de respuesta	Average response time
Tiempo medio entre fallos	Mean time to failure
Tiempo medio entre pérdidas de datos	Mean time to data loss
Tiempo para finalizar	Time to completion
Tiempo válido	Valid time
Tipo colección	Collection type
Tipo de conexión ficticia	Dummy junction type
Tipo de enlace ficticio	Dummy link type
Tipo de registro ficticio	Dummy record type
Tipo de registro Renlace	Record type Rlink
Tipo más específico	Most specific type
Tipos complejos	Complex types
Tipos de dominios	Domain types
Tipos de referencia	Reference types
Tipos estrictos	Strong types
Tipos no atómicos	Nonatomic types
Tolerancia ante fallos	Fault tolerance
Total	Total
Trama	Mesh
Transacción	Transaction
Transacción anidada	Nested transaction
Transacción compensadora	Compensating transaction
Transacción comprometida	Commited transaction
Transacción de actualización	Update transaction
Transacción de alto rendimiento	High-performance transaction

Transacción de corta duración	Short-duration transaction
Transacción de larga duración	Long-duration transaction
Transacción dudosa	In-doubt transaction
Transacción global	Global transaction
Transacción interactiva compleja	Complex interactive transaction
Transacción local	Local transaction
Transacción multinivel	Multilevel transaction
Transacción por minilotes	Mini-batch transaction
Transacciones anidadas	Nested transactions
Transacciones por segundo	Transactions per second (TPS)
Transferir	Transfer
Transitorio	Transient
Transparencia de la red	Network transparency
Tratamiento de interbloqueos	Deadlock handling
Tries	Tries
Trivial	Trivial
Tupla	Tuple
Turno rotatorio	Round-robin
UCP (Unidad central de procesamiento)	Central processing unit (CPU)
Unaria	Unary
Único	Unique
Unidad	Unit
Unidad central de procesamiento (UCP)	Central processing unit (CPU)
Unión	Union
Univalorados	Single valued
Uno seguro	One-safe
Usuario	User
Usuario normal	Naive user
Utilidades para presentaciones	Presentation facilities
Utilización	Utilization
Utilización de elementos de datos	Data-item utilization
Utilizado más recientemente	Most recently used (MRU)
Utilizado menos recientemente	Least recently used (LRU)
Valores de salida	Output values
Valores nulos	Null values
Variable	Variable
Variable de dominio	Domain variable
Variable de ejemplares	Instance variable
Variable externa	External variable
Variable ligada	Bound variable
Variable tupla	Tuple variable
Varias transacciones	Multiple transactions

VDD (Videodisco digital)	Digital video disk (DVD)
Vector de división	Vector partitioning
Vector de división con carga equilibrada	Load-balanced partition vector
Vector de versiones	Version vector
Velocidad de transferencia	Transfer rate
Velocidad de transferencia de datos	Data-transfer rate
Versiones multiconjunto	Multiset versions
Videodisco digital (VDD)	Digital video disk (DVD)
Violación de la segmentación	Segmentation violation
Vista	View
Vista materializada	Materialized view
Vistas equivalentes	Equivalent views
Visualización de datos	Data visualization
Visualizar	Display
Volcado	Dump
Volcado de archivo	Archival dump
Volcado difuso	Fuzzy dump
Volcar	Dump
Volumen	Volume
Volver a configurar	Reconfigure
Web crawler	Web crawler
World Wide Web	World Wide Web (WWW)
Zeta	Theta (θ)
Zona	Area

ÍNDICE

FUNDAMENTOS DE BASES DE DATOS

CUARTA EDICIÓN

SILBERSCHATZ • KORTH • SUDARSHAN

Prepárese para afrontar el mundo real de las aplicaciones de bases de datos y su implementación. Esta cuarta edición completamente revisada presenta los conceptos fundamentales de la gestión de bases de datos, incluyendo el diseño de bases de datos, lenguajes y la implementación del sistema. Todos los conceptos se presentan de forma intuitiva con figuras claras y bien estructuradas, y ejemplos que relevan a las pruebas formales. Un ejemplo bancario ilustra los conceptos que se van introduciendo.

Esta cuarta edición contiene:

- ♦ Tratamiento extendido del modelo ER, incluyendo la notación UML.
- ♦ Tratamiento extendido de SQL, incluyendo SQL: 1999, ODBC y JDBC.
- ♦ Apartado sobre QBE, incluyendo la implementación QBE de Access de Microsoft.
- ♦ Tratamiento mejorado y más fácil de comprender de la normalización.
- ♦ Tratamiento actualizado de los modelos: orientado a objetos y relacional orientado a objetos.
- ♦ Nuevo capítulo sobre XML.
- ♦ Materiales adicionales sobre el procesamiento de consultas, incluyendo las vistas materializadas.
- ♦ Nuevo capítulo sobre administración de bases de datos.
- ♦ Nuevo capítulo sobre aspectos del desarrollo de aplicaciones, incluyendo el acceso Web a las bases de datos.
- ♦ Tratamiento extendido de los almacenes de datos (*data warehouse*), OLAP y recopilación de datos (*data mining*).
- ♦ Tratamiento actualizado y extendido de la recuperación de información.
- ♦ Nuevos apéndices sobre el diseño interno de los sistemas de bases de datos comerciales.

http://www.mcgraw-hill.es/olc/silberschatz

LOS AUTORES

Abraham Silberschatz (Dr. por la Universidad estatal de Nueva York en Stony Brook) es vicepresidente del Centro de investigación de ciencias de la información en Bell Labs, Lucent Technologies, Inc. Sus intereses en investigación incluyen los sistemas operativos, los sistemas de bases de datos y los sistemas distribuidos. El Dr. Silberschatz ha sido miembro del Panel de biodiversidad y ecosistemas para el Comité de consejeros de ciencia y tecnología del presidente Clinton, y consejero de la Fundación nacional de ciencias (NSF). Ha recibido en 1998 el premio al mejor educador Karl V. Karlstrom de la sociedad ACM, el premio a la contribución a SIGMOD de ACM y el premio por el mejor artículo de la Sociedad informática IEEE por su artículo "Gestor de capacidades". El Dr. Silberschatz es miembro de las sociedades ACM e IEEE.

Henry F. Korth (Dr. por la Universidad de Princeton) es director de investigación en principios de bases de datos de Bell Labs, Lucent Technologies, Inc. También es Director de evaluación de tecnologías del Grupo de productos software de Lucent Technologies. Entre sus líneas de investigación se encuentran los sistemas de bases de datos de alto rendimiento, sistemas de bases de datos de tiempo real y bases de datos XML, el procesamiento de transacciones, la informática portátil, los sistemas de bases de datos en tiempo real y el procesamiento paralelo y distribuido. El Dr. Korth es miembro de la sociedad ACM y miembro senior de IEEE. Ha escrito del orden de 100 artículos técnicos y ha sido miembro de los paneles editoriales de varias revistas informáticas. Actualmente es miembro de los paneles de *The VLDB Journal* y *The International Journal on Cooperative Information Systems*.

S. Sudarshan (Dr. por la Universidad de Wisconsin, Madison) es profesor asociado en el Departamento de informática e ingeniería del Instituto de tecnología indio en Bombay. Anteriormente ha sido miembro del equipo técnico de los Laboratorios AT&T Bell (ahora Lucent Technologies). Entre las líneas de investigación del Dr. Sudarshan se encuentran el procesamiento y optimización de consultas, recuperación de fallos y bases de datos en memoria principal.



http://www.mcgraw-hill.es

McGraw-Hill Interamericana de España, S. A. U.
A Subsidiary of The McGraw-Hill Companies

ISBN: 84-481-3654-3